Syllabus

Syllabus

- Download Syllabus (PDF)

Course Information

- CSC210, Spring 2026
- Meeting time: Tuesday/Thursday | 10:50 AM – 12:05 PM
- Location: Ford 342
- Primary Instructor: Prof. Halie Rando (she/her) (Prof. Rando, Dr. Rando, Halie, or Prof. Halie)
- Email: hrando@smith.edu
- Office: Bass Hall 109
- Office hours: Monday 10:00 – 11:00 (Bass 109), Wednesday 11:00 – 12:00 (Bass 109), Friday 2:00 – 3:00 (Bass 109; Getting Started / Design Questions Only)

Welcome to CSC210: Data Structures! We hope you will have fun in this course while learning a lot about data structures, which are fundamental concepts in computer science. Below is some important information about how the course will run, as well as some specific expectation-setting details about course norms and what we will and won't cover.

Course Description

In this course, we will explore elementary data structures (arrays, sequences, linked lists, stacks, queues, trees, graphs) and algorithms (searching, sorting) in a variety of contexts. The primary goal is that you become deeply familiar with these structures and algorithms and learn their relative strengths and weaknesses, as well as the pros and cons of different implementations. The secondary goal is to build your skills and confidence to design and build robust computational solutions as you embark into future courses, challenges, and/or careers in computer science.

Prerequisite This course requires CSC120 because it assumes, and requires, that you have a sufficient understanding of object-oriented programming and

Java programming to implement your ideas. We will be working with the Java programming language, but data structures themselves are important no matter what your preferred language is, and the concepts you will learn this semester are broadly applicable across languages.

What We Will Cover

This course will provide a thorough overview of the foundations of data structures and is scoped to a target audience of CSC majors who have completed the first two courses in programming at Smith. If you have arrived in this class through some other path, don't worry! You're still very much welcome in this course. Just be mindful that some self-study before the course starts may be necessary (see What We WON'T Cover). Throughout the course we will reinforce skills you should have learned in previous courses, such as object-oriented programming in Java.

What We WON'T Cover

This course assumes that you have a solid foundation in object-oriented programming, either through successful completion of CSC120, multiple courses at the high school level, or dedicated self-study. If you do not feel comfortable writing code in Java, you should spend time on your own learning the basics of the language and how it may differ from others you have learned. This is not a Java class. If applicable, you may want to consider taking CSC 120 and coming back to this class next semester.

Proficiency with various computer environments and infrastructure will also come in handy. Because folks are coming to this class from a variety of backgrounds, we strongly recommend paging through The Missing Semester of Your CS Education to familiarize yourself with topics such as: - Shell Tools and Scripting - Editors (Vim) - Command-line Environment - Version Control (Git) - Debugging and Profiling

Additionally, knowledge of the following will likely prove useful in this class, but due to time constraints we will unfortunately not be covering the following topics (links to supplemental online courses through LinkedIn Learning are provided, which are available for free to all members of the Smith community): - Programming Foundations: Fundamentals (Been awhile since you took 110? This 2hr crash course will remind you of the key points. Taught in Python.) - Learning Java (Worried about making the leap from Python to Java? This course is essentially a do-over of 110/120, but in Java instead of Python.) - Java Code Challenges (Partway through the semester and you still feel like you just don't quite "get" Java? Try these challenges to build your confidence!)

Learning Goals

This course is designed to help you develop a toolkit for solving computational problems and to approach programming with flexibility and resilience. We

organize this study around three main themes. Upon completing this course, students will:

Theme 1: Understanding Data Structures and Their Abstractions - Describe fundamental data structures in terms of both the operations they support and the associated conceptual model. - Map abstract data types to potential implementations, both concrete and abstract. - Reason about strengths and weaknesses of a particular implementation.

Theme 2: Knowing How to Apply Data Structures - Solve problems computationally through the application of fundamental data structures and algorithms. - Choose an appropriate data structure for a given task and justify the choice (considering constraints including time, space, etc.) - Compare multiple proposed solutions to a given problem and evaluate their relative merits.

Theme 3: Implementing Algorithms and Using Appropriate Data Structures to Solve Programming Problems - Write clear, well-documented, and effective programs using techniques such as recursion, object-oriented design, and functional programming. - Demonstrate comprehension of written code (e.g., tracing execution, debugging, etc.). - Implement a variety of algorithms for solving computational problems (e.g., sorting, search, etc.), and articulate the pros and cons of different methods.

Course Format

While programming can be a truly interesting, challenging and rewarding intellectual activity, it is almost impossible to learn it without a lot of practice. We want to devote as much class time as possible to actual programming with feedback from professors, teaching assistants, and peers.

Day-to-Day Structure

Most classes will be hands-on and interactive. Occasionally, you may be asked to watch a short video or complete a brief reading ahead of class to support in-class work or an assignment. Here are some useful tips to help orient you to this model (adapted from UC Boulder's Succeeding in a Flipped Classroom): 1. Expect work to be continuous. Rather than cramming for large exams that happen after classes, this course requires you to keep up with frequent assignments, readings, videos and quizzes that are due before class. One way to set yourself up for success is to get into the habit of completing your pre-class tasks at the beginning of the semester. Keep up with the course website for updates and assignments. 2. Show up present and prepared. Be present in class, physically and mentally, to the best of your ability. Be ready engage in class discussions and activities. Ask questions to clarify your understanding of concepts, offer your own perspective, and try not to be afraid of giving "wrong" answers misconceptions and false starts are a normal, healthy part of learning (and we professors guarantee we'll make plenty of our own mistakes for you to catch!) 3. Use prerecorded lectures and readings to your advantage. While watching recorded lectures may seem

like a pain, they provide opportunities to learn at your own pace. If you benefit from rewatching videos or pausing to try something out on your own, please do! Many students find it helpful to take notes while watching prerecorded lectures, just like they would in a traditional class. 4. If you find yourself spending inordinate time debugging your programs without really making progress, it's time to reach out before you fall behind, or it affects your work in other courses. Your professors and TAs are eager to help you find strategies that work for you, and that enable you to reach your goals in the course. In addition to us, the college has many resources for academic success, all available at no cost.

Course Materials

There is no required textbook for this course: any required readings or videos will be made available through the course website (and occasionally Moodle), and primary content delivery will come via Powerpoint slide decks. However, there are a few recommended textbooks for those who find them useful as references. None of these textbooks are a perfect fit for the class, but we have drawn material from many of them and all are available for free online, or you can purchase a hard copy from your favorite bookstore: - Open Data Structures (in Java) (Pat Morin), Athabasca University Press, 2013. Offered as an introduction to the field of data structures and algorithms, Open Data Structures covers the implementation and analysis of data structures for sequences (lists), queues, priority queues, unordered dictionaries, ordered dictionaries, and graphs. Focusing on a mathematically rigorous approach that is fast, practical, and efficient, Morin clearly and briskly presents instruction along with source code. - Think Java: How to Think Like a Computer Scientist, 2nd Ed. (Allen Downey and Chris Mayfield.), O'Reilly, 2019. Think Java is a hands-on introduction to computer science and programming used by many universities and high schools around the world. Its conciseness, emphasis on vocabulary, and informal tone make it particularly appealing for readers with little or no experience. The book starts with the most basic programming concepts and gradually works its way to advanced object-oriented techniques. In this fully updated and expanded edition, authors Allen Downey and Chris Mayfield introduce programming as a means for solving interesting problems. Each chapter presents material for one week of a college course and includes exercises to help you practice what you've learned. If you need help covering the cost of textbooks or other academic supplies (for this or any of your courses!) please fill out the Academic Funding Application.

Communication & Response Norms

The course website is the source of truth for policies, deadlines, and updates. Please use the channels below to keep questions organized and response times predictable.

Primary channels

- GitHub Issues (or PRs) are preferred for assignment questions and clarifications.
- Slack is for general course questions, peer help, and TA support. I do not monitor Slack continuously.
- Email is for private or time-sensitive matters.
- Moodle may be used occasionally for submissions or feedback, but Gradescope is the default for submissions.

Slack expectations

- Slack is asynchronous, not real-time chat.
- Slack is primarily for students to communicate with and support each other, though the teaching staff tries to check the public channels at least once per day.
- Responses may take time, especially outside business hours.
- I do not monitor Slack continuously or outside business hours.
- Slack DMs are strongly discouraged; use public channels instead.
- Office hour changes will be posted in the `#announcements` channel.
- Feel free to post playfully relevant content in `#random`.

Email expectations

- Email is checked once per business day.
- You can expect a response within 24–48 business hours.
- Emails sent outside business hours are treated as arriving the next business day.
- Emails about CSC210 should use the subject line format `CSC210: <brief description>`.
- (Use the course email template to help me respond quickly)[mailto:hrando@smith.edu?subject=CSC210:%20

How to ask for help effectively

- Put your full question in one message.
- Include context (assignment name, deadline, and what you tried).
- Avoid sending multiple follow-ups before receiving a reply.
- Emails that do not use the required subject format and template may receive delayed responses or be redirected to office hours.

Requests that require advance notice

Some requests (extensions, recommendation letters, special arrangements) require advance notice and may have additional requirements. Please consult the course website for details.

Office hour cancellations

If office hours are cancelled, students may attend any remaining office hours that week or use email to ask brief questions. No student will be penalized for missing office hours due to a cancellation.

Friday office hours (scope)

Friday office hours focus on scoping, design, and early debugging for the next assignment. Deeper debugging and final checks are best handled during Monday/Wednesday office hours. If Friday is your only option, come anyway and we'll triage. All students are welcome in office hours, and I often have candy.

Assessment

There are five forms of assessment in this course: - Weekly programming homework assignments - Weekly prototyping assignments - Midterms - In-class activities - Self-scheduled final exam

Exams

Midterm exams will be arranged through the Science Center. You can find more information about Science Center self-scheduled exams on the Science Center webpage. The final exam will be held during the self-scheduled final exams period. More information about final exams is available through the registrar.

This semester we will have three midterms. Each midterm adds new topics, and also includes all previous topics available for retakes. - Midterm 1, Feb 20–22: 6 new topics. - Midterm 2, Mar 27–29: 5 new topics + all topics from Midterm 1 available for retakes. - Midterm 3, Apr 24–26: 5 new topics + all topics from Midterms 1–2 available for retakes. - Final (optional): all topics from Midterms 1–3 available for retakes, no new topics.

Each topic refers to a specific conceptual data structure or algorithm (e.g., linked lists, recursion, trees). Each topic is scored based on the highest score you earned on that topic across all exams; earlier lower scores are not averaged in. This means one bad day will not hurt you, and you can raise your score by mastering material later in the semester. While scores are replaced by your best performance, later exams may still rely on earlier topics as prerequisites. Pedagogically, we consider 90%+ mastery, so you are encouraged to prioritize retakes for topics where you scored below 90%. For example, if you score 70% on Topic A on Midterm 1, 55% on Topic A on Midterm 2, and 95% on Midterm 3, your Topic A score is 95%.

Homework and Labs

Weekly programming assignments will be introduced in class on Thursday, and we will work together to get started on them during class time. You will be

provided with a rubric in the form of a checklist each week. Assignment deadlines are Tuesdays at 11:59PM ET. TA hours are an especially good place to discuss prototype slides, design decisions, and early implementation plans before you begin coding.

Compilation requirement Code must compile to be graded. Submissions that do not compile may receive a zero or be returned ungraded, and the autograder's compilation result is authoritative. The autograder environment is the reference environment for this course; local execution does not override autograder results. This is a gate condition: 100% of an assignment's points depend on successful compilation. For example, code that fails to compile due to missing methods, syntax errors, or incorrect signatures cannot be graded, even if portions are correct.

Regrades You are invited to submit homework assignments for regrading if you receive <90%. These resubmissions are due on the last day of classes at 11:59PM. You are welcome to modify your resubmission at any time before the end of the semester, but regrading is completed only at the end of the semester, not on a rolling basis. However, you are welcome to come discuss past assignments in office hours. Regraded scores are capped at a maximum of 90%, even if the resubmission is perfect. You should submit your assignment with an updated checklist, noting what changed and request a regrade in Gradescope.

Extension Policy 72-hour extensions are automatically approved when submitted ahead of the deadline at the extension request form.

Please note: - Up to 72 hours is granted automatically through the form. - Extensions set a clear, final submission date. - The extension deadline functions exactly like the original deadline; submissions after that time cannot be accepted. - The only exception is documented extenuating circumstances communicated by the Class Deans' Office. In rare cases of serious, documented circumstances, the Class Deans' Office may contact the instructor to coordinate next steps. - If you need more than 72 hours due to an extreme circumstance, email before the original deadline. - Students are strongly encouraged to submit what they have early; autograder feedback often helps with debugging. - Assignments not submitted by the original (or extended, if requested) deadline are not eligible for regrading.

Attendance and Participation

This is not your average data structures class – we will have extensive group work and hands-on activities that it will be hard to catch up on if you miss class. **Class attendance is required** unless you have let the professor know about your absence (e.g., for illness, conference travel, etc.) via email. In-class lab activities are designed to be completed during the scheduled class time in which they are assigned and may be difficult or impossible to replicate independently.

Students who are present and participate in class receive full credit for the work. If you have an unexcused absence, you are responsible for making up any missed activities within one week, by coordinating with the instructor or TAs, preferably before the next class. Bear in mind that the lab is often a warmup for an upcoming assignment; labs should not interfere with beginning your work on the programming assignment. You can take notes however you want while information is being presented, although you are encouraged to try using a pen and paper to see if it is helpful to your learning. However, at some points, you will be asked to "clamshell" (close) your computer to work on a project hands-on or to sketch out ideas for an implementation. During these exercises, you will be asked to collaborate with other students, so there will not be a requirement that all students write. If you need any additional accommodations related to challenges in this area, please reach out to Prof. Rando or to the Accessibility Resources Center (ARC; see below) and we can discuss how to adjust it.

Collaboration and Academic Integrity

Collaboration during in-class labs may involve shared work; graded homework assignments must be completed individually. You may discuss approaches, algorithms, and design decisions with classmates, but you must write, debug, and submit your own code. You may not view another student's code before submitting your own work. Students are strongly encouraged to form study groups and to collaborate in solving the assignments. You should include a list of all students you worked with in your README file for each assignment. Please ensure that all work you submit is ultimately the product of your own understanding rather than anyone else's.

Peer support norms

In this course, programming assignments are designed to help you build durable, individual understanding. Collaboration is encouraged at the level of ideas and design, and we strongly encourage pre-coding peer review (before you start writing code). Writing code is an individual activity so that you can accurately assess your own understanding before exams.

Students may: - Explain their design to a peer. - Walk through invariants. - Talk through edge cases.

Students may not: - Share code. - Screenshare code. - Paste snippets. You may consult online or print references on all assignments and labs. Standard language references showing syntax, usage, class Javadoc, etc. need not be cited. All other resources must be cited as described below. The following information is required for all submitted work: 1. The names of all collaborating students be listed at the top of the submission (in your README file). We invite you to give "kudos" to students who were particularly helpful in your learning. 2. A "References" section, with in-line citations to any external resources you used. Citations should include page numbers (if a printed resource) or a direct URL (if an online

resource). If you did not use any resources in completing the assignment, please state: "I did not utilize any external resources in completing this assignment." If you include a fragment of code from any source, you should also credit that source with a comment directly in the code. 3. If you use any AI programming assistants, you should also cite this at the top of your README file, along a description of the type of assistance provided and how it intersected with your learning process. You are responsible for being able to explain and reproduce any code you submit without assistance. Citing a tool does not necessarily make its use appropriate; all submitted work must reflect your own understanding. Using AI to diagnose or fix bugs is considered code assistance and should be cited. Please keep in mind that using AI help is similar to copying someone else's code or asking another person for help: you may achieve more, but you may learn less unless you make an effort to build on what is shared. We hope that you are in this course for the learning, and thus discourage use of AI prompting in most situations. Your grade will be based upon the value that you add, not on content from any other sources.

Academic integrity process

If I have concerns about academic integrity, I am required to meet with the student to share those concerns and then refer the matter to the Academic Integrity Board to manage next steps. I do not adjudicate whether or not a violation occurred. More information is available here: https://www.smith.edu/your-campus/offices-services/dean-college/academic-integrity-board.

TA Hours at the Spinelli Center

Students who succeed in CSC210 routinely set aside time to attend TA hours, even when they are not stuck. TA hours are where most students clarify design choices, validate their approach, and catch small mistakes before they become large ones. Students in this class are expected to make use of the CS TA Help Center as part of their regular workflow. By default, CS TAs will discuss individual student issues and general class troubles with the course instructor. However, if you would like to attend TA hours confidentially, please let the TA know at every session. You are expected to collaborate with peers and TAs in these sessions. The use of generative AI is not permitted in TA hours. Your peers and TAs are a better resource – talk to them instead. TA hours info: https://www.smith.edu/academics/integrative-learning/spinelli-center-quantitative-learning/tutoring-hours#comp_sci. Peer support norms apply during TA hours as well (see "Peer support norms" above): discuss ideas and design, but do not share code or snippets.

Grading

Grades reflect demonstrated understanding, correctness, and engagement with the assignment's learning goals. Grades do not reflect ambition level, career

direction, or over-engineering beyond the learning goals of assignments.

Your final grade is based on our evaluation of your work, and every effort will be made to communicate expectations in advance through detailed rubrics. Although the grade will be largely based on the percentages shown below, we reserve the right to award extra credit for excellent work and out-of-the-box thinking. For example, while "Participation and Engagement" will look primarily at day-to-day engagement, we will also take note of contributions both in and out of class which demonstrate intellectual curiosity or clear understanding of a topic, as well as effort to help others learn difficult concepts in office hours, TA hours, etc. Participation on the class slack forum and opening GitHub issues or pull requests to help improve assignments is especially encouraged.

The Final Project (Computer Science Fair presentation) is a hands-on activity about an advanced topic relevant to the course. There are checkpoints for a topic submission, outline, and handout before the final presentation. Students will present their activity at the hands-on fair on April 23.

Category	Percentage
Coding/Written Assignments	55%
Final Project	5%
Exams	30%
Participation and Engagement	10%

Accessibility

We aim to make this course accessible and welcoming to all. Please let us know if there are any changes we can make towards meeting this goal. Please avoid wearing heavily scented products (e.g., perfume, cologne) to class, as fragrances can cause health issues for some community members, including faculty with offices near our classroom. Smith College can also help facilitate support services and accommodations to all students with disabilities. To request an accommodation, please register with the Accessibility Resources Center (formerly ODS).

Authoritative Sources

If there is a discrepancy in course information, the following order of authority applies: autograder behavior, assignment PDF, course website, and then other explanations. The course website remains the source of truth for schedules and policies, but assignment specs and the autograder control assignment interpretation.

Acknowledgement

Some of the materials used in this course are derived from lectures, notes, or similar courses taught at other institutions. Appropriate references will be included on all such material. Materials (including this syllabus and many of the slide decks!) are heavily influenced by faculty at Smith who have taught this material previously, especially Prof. Jordan Crouser and Prof. Nick Howe – thank you both! And kudos to Prof. Alicia Grubb for the extension policy.