ECE 2400 Computer Systems Programming, Fall 2017

Course Syllabus

School of Electrical and Computer Engineering
Cornell University

revision: 2017-08-24-16-52

1. Course Information

Cross Listed	ENRGD 2140 Computer Systems Programming
Prereqs	CS 1110 (preferred) or CS 1112
Instructor	Prof. Christopher Batten, 323 Rhodes Hall, `cbatten@cornell.edu` Office Hours: 323 Rhodes Hall, Tuesday, 4:30–5:30pm
Admin. Assistant	Daniel Richter, 314 Rhodes Hall, `tdr27@cornell.edu`
Graduate TAs	Christopher Torng `clt67` Office/Lab Hours: 314 Phillips, Mon, 7:30–9:30pm
Undergraduate TAs	Alex Katz `aik49` Office/Lab Hours: 314 Phillips, Mon, 7:30–9:30pm Juan Albrecht `ja643` Office/Lab Hours: 314 Phillips, Thu, 7:30–9:30pm
Lectures	206 Upson Hall, Mon/Wed/Fri, 10:10–11:00am
Disc. Section	403 Phillips Hall, Friday, 2:30–3:20pm
Required Materials	A. Hilton and A. Bracy, "All of Programming," 2015 Only available as an ebook ($10) `http://aop.cs.cornell.edu`
Website	`http://www.csl.cornell.edu/courses/ece2400`
Staff Email	`ece2400-staff-l@cornell.edu`

2. Description

Computer systems programming involves developing software to connect the low-level computer hardware to high-level, user-facing application software and usually requires careful consideration of performance and resource constraints. Examples of computer systems software include compilers, operating systems, databases, numerical libraries, and embedded controllers.

This course aims to provide a strong foundation in computer systems programming using C and C++, the languages of choice for system-level programmers. The course is structured into three parts. In the first part, students will learn basic C programming (e.g., static typing, functions, control flow, arrays, strings, pointers, dynamic memory management); students will then apply their knowledge of C to explore basic algorithmic techniques (e.g., recursion, divide-and-conquer, dynamic programming), basic algorithms (e.g., sorting), and basic data structures (e.g., lists, stacks, queues, sets, maps). In the second part, students will learn more advanced C++ programming (e.g., objects, inheritance, polymorphism, templates); students will then apply their knowledge of C++ to explore more advanced algorithms and data structures (e.g., binary search trees, priority queues,

hash tables, graphs, spanning trees). In the third part, students will explore systems programming in the UNIX environment using POSIX I/O, processes, and threads.

The course includes a strong emphasis on both theoretical and practical design trade-offs. A series of programming assignments enable students to apply what they have learned to interesting real-world problems and to gain experience with version control, continuous integration, debugging, profiling, and code optimization.

3. Objectives

This course is meant to be a foundational course in computer systems programming. The course will prepare students for more advanced coursework in computer engineering (e.g., embedded systems, computer architecture) as well as more advanced coursework that focuses on a single type of computer systems software (e.g., compilers, operating systems, databases). By the end of this course, students should be able to:

- **demonstrate** the ability to effectively use the C and C++ programming languages to solve small programming problems.
- **describe** both basic and advanced algorithms and data structures, how to implement these algorithms and data structures in C/C++, how to analyze algorithms using computational cost, and how to use POSIX I/O, processes, and threads for systems programming in the UNIX environment.
- **apply** this understanding to solve new programming problems.
- **evaluate** various algorithm and data structure alternatives and make a compelling qualitative and/or quantitative argument for why one approach is superior.
- **create** non-trivial C/C++ programs (roughly 1000 lines of code) and the associated testing strategy starting from an English language specification.
- **write** concise yet comprehensive technical reports that describe a program implemented in C/C++, explain the testing strategy used to verify functionality, and evaluate the program to characterize its performance and memory usage.

4. Prerequisites

This course is targeted towards sophomore-level undergraduate students, although it is also appropriate for advanced freshman students and upperclassman. An introductory course on computing is required. Students need to be comfortable using at least one programming language (e.g., Python through CS 1110 or MATLAB through CS 1112) and should have some experience in software design, development, and testing. No prior knowledge of the C or C++ programming languages is necessary.

5. Topics

The course includes three parts. The first part covers C programming and then uses C to explore basic data structures and algorithms, while the second part covers C++ programming and then uses C++ to explore more advanced data structures and algorithms. The third part covers systems programming in the UNIX environment. A tentative list of topics for each part is included below. The exact topics covered in the course are subject to change based on student progress and interest.

- **Part 1: C Programming and Basic Data Structures & Algorithms –** variables, expressions, functions, control flow, recursion, types, pointers, arrays, dynamic allocation, computational cost, abstract data types, lists, stacks, queues, sets, maps, sequence sorting, $O(n^2)$ sorting algorithms, $O(n\ log(n))$ sorting algorithms, sequence alignment, dynamic programming
- **Part 2: C++ Programming and Advanced Data Structures & Algorithms –** transition from C to C++, namespaces, references, function overloading, object-oriented programming, template meta-programming, polymorphism, binary search trees, priority queues, hash tables, graphs
- **Part 3: Systems Programming in the UNIX Environment –** standard I/O, processes, threads, concurrency, synchronization

6. Required Materials

The required textbook for the course is *"All of Programming,"* by A. Hilton and A. Bracy (2015). The book costs $10 and is only available as an ebook through the Google Play Store. The book takes advantage of the ebook format to include over seven hours of embedded videos. To learn more about how to purchase and read the book use this link: `http://www.csl.cornell.edu/courses/ece2400/readings`.

7. Optional Materials

There are a several additional books that students may find useful for providing additional background or more advanced material. All of these books are on reserve at Uris Library, and many of these books are also availbale as ebooks through the Cornell library.

- K.N. King. *"C Programming: A Modern Approach,"* 2nd edition. W.W. Norton & Co., 2008.
- R. Reese. *"Understanding and Using C Pointers,"* 1st edition. O'Reilly Media, 2013.
- R. Sedgewick. *"Algorithms in C, Parts 1–4,"* 3rd edition. Addison-Wesley, 1998.
- S.B. Lippman, J. Lajoie, and B.E. Moo. *"C++ Primer. Addison-Wesley,"* 5th edition. Addison-Wesley, 2013.
- M. Weiss. *"Data Structures and Algorithm Analysis in C++,"* 4th edition. Pearson, 2013.
- W.R. Stevens and S.A. Rago. *"Advanced Programming in the UNIX Environment,"* 3rd edition. Addison-Wesley, 2013.
- A. Williams. *"C++ Concurrency in Action: Practical Multithreading,"* 1st edition. Manning Pub, 2012.

8. Format and Procedures

This course includes a combination of lectures, short in-class quizzes, optional discussion sections, assigned readings, programming assignments, and exams.

- **Lectures –** Lectures will be from 10:10am to 11:00am every Monday, Wednesday, and Thursday in 206 Upson Hall excluding the following academic holidays: Labor Day (9/4), Indigenous

People's Day (10/9), and Thanksgiving (11/22). We will start promptly at 10:10am so please arrive on time. Students are expected to attend all lectures, be attentive during lecture, and participate in class discussion. Please turn off all cellular phones during class. Use of cellular phones and laptops during lecture is not allowed (see Section 12.C).

- **Quizzes –** There will be a short quiz at the beginning of some lectures. The quiz should take about five minutes, and will cover some of the key topics discussed in the previous lecture. Quizzes are not announced ahead of time, and there are no make-up quizzes. The lowest quiz score is dropped which effectively provides for one excused absence. Students should prepare for a potential quiz by simply reviewing the material from the previous lecture before coming to class. Solutions to quizzes will be available online soon after the quiz is given for formative self-assessment.
- **Discussion Section –** The discussion section will be most Fridays from 2:30pm to 3:20pm in 403 Phillips Hall. Attendance at the weekly discussion sections is optional but strongly encouraged. These discussion sections will be relatively informal, with the primary focus being on facilitating student's ability to complete the programming assignments and on reviewing material from lecture using problem-based learning.
- **Readings –** Students are expected to complete all of the required reading according to the schedule on the course website, although there is some flexibility. Some students may prefer to complete the readings before the corresponding lecture, while others may prefer to complete the readings after the corresponding lecture. Either strategy is acceptable. All of the required readings are contained within the course textbook.
- **Programming Assignments –** The course will include six programming assignments. Students are expected to work individually on the first four programming assignments and in a group of two students for the final two programming assignments. Students will be using the ECE Computing Resources to complete the programming assignments, the code must be submitted via GitHub, and the report must be submitted in PDF format via the online CMS assignment submission system (see Section 13). No other means of submission will be accepted. Programming assignments are due on Tuesdays at 11:59pm except for the final programming assignment which is due on a Thursday at 11:59pm (see Section 12.D for late assignment policy).
- **Prelim and Final Exams –** The course includes two prelim exams and a cummulative final exam. If students have a scheduling conflict with the exam, they must let the instructor know as soon as possible, but no later than two weeks before the prelim or final exam. Graded final exams and the exam solutions are only available for review under the supervision of a course instructor. You may not remove your graded exam, nor may you remove the exam solutions.

9. Assignment and Exam Schedule

The current schedule is on the course website. All programming assignments are due on Tuesdays at 11:59pm except for the final programming assignment which is due on a Thursday at 11:59pm. Changes to this schedule will be posted as announcements via Piazza.

Tue	Sep	12	PA1.1 – Complex Math Functions
Tue	Sep	19	PA1.2 – Complex Math Functions
Tue	Oct	3	PA2 – Cracking Passwords
Tue	Oct	5	Prelim from 7:30–10:30pm in 101 Phillips Hall
Tue	Oct	17	PA3.1 – Searching and Sorting
Tue	Oct	24	PA3.2 – Searching and Sorting
Tue	Oct	32	PA4 – RPN Calculator
Tue	Nov	9	Prelim from 7:30–10:30pm in 101 Phillips Hall
Tue	Nov	14	PA5 – In-Memory Cache for Distributed Systems
Thu	Nov	30	PA6 – Financial Trading System
Sat	Dec	9	Final Exam from 2:00-5:00pm (location TBD)

10. Grading Scheme

Each part or criteria of every assignment is graded on a four-point scale. A score of 4.25 is an A+, 4 roughly corresponds to an A, 3 roughly corresponds to a B, 2 roughly corresponds to a C, and so on. A score of 4.0 usually indicates that the submitted work demonstrates no misunderstanding (there may be small mistakes, but these mistakes do not indicate a misunderstanding) or there may be a very small misunderstanding that is vastly outweighed by the demonstrated understanding. A score of 3.0 usually indicates that the submitted work demonstrates more understanding than misunderstanding. A score of 2.0 usually indicates that the submitted work demonstrates a balanced amount of understanding vs. misunderstanding. A score of 1.0 usually indicates that the submitted work demonstrates more misunderstanding than understanding. A score of 4.25 is reserved for when the submitted work is perfect with absolutely no mistakes or is exceptional in some other way.

Total scores are a weighted average of the scores for each part or criteria. Parts or criteria are usually structured to assess a student's understanding according to four kinds of knowledge: basic recall of previously seen concepts, applying concepts in new situations, qualitatively and quantitatively evaluating alternatives, and creatively implementing new designs; these are ordered in increasing sophistication and thus increasing weight. In almost all cases, scores are awarded for demonstrating understanding and not for effort. Detailed rubrics for all quizzes, programming assignments, and exams are provided once the assignment has been graded to enable students to easily see how the score was awarded. A detailed Programming Assignment Assessment Rubric is available on the public course webpage.

The final grade is calculated using a weighted average of all assignments. All quiz grades are averaged to form a single total. Students can drop their lowest quiz score. At the instructor's discretion, additional quiz scores may be dropped depending on the total number quizzes in the semester and pseudo-quiz grades may be used to encourage participation, completing student evaluations, etc. The weighting for the various assignments is shown below.

Quizzes	10%	(students can drop lowest score)
PA1–4	20%	(weighted equally)
PA5–6	15%	(weighted equally)
Prelim Exams	30%	(weighted equally)
Final Exam	25%	

Note that the exams account for over half of a student's final grade. The exams in this course are very challenging. Successful students begin preparing for the exams far in advance by carefully reviewing

the assigned readings, independently developing study problems, and participating in critical study groups.

To pass the course, a student must at a bare minimum satisfy the following requirements: (1) submit five out of the six programming assignments; (2) take both prelim exams; and (3) take the final exam. **If a student does not satisfy these criteria then that student may fail the course regardless of the student's numerical grade.**

11. Degree Requirements

This section describes how ECE 2400 can satisfy various degree requirements. Students are responsible for confirming this information with the appropriate student services coordinator.

- **ECE Undergraduate Students –** ECE 2400 can count as your second engineering distribution course (all ECE students must take ECE 2300 as their first engineering distribution course). Alternatively, all ECE students can count this course as an outside-ECE technical elective. Students are reminded they are only allowed to count one ECE course as an outside technical elective. Regardless, this course satisfies the ECE advanced programming requirement.
- **ECE M.Eng. Students –** ECE 2400 can count as a technical elective course, although students are reminded they are only allowed to count one ECE undergraduate course as a technical elective.
- **CS Undergraduate Students –** ECE 2400 may be able to count as your second engineering distribution course. However, since all CS students must take CS 2110 as their first engineering distribution course, taking both CS 2110 and ECE 2400 may not be in the student's best interest. CS majors should probably speak with the instructor to ensure ECE 2400 meets their overall educational goals.
- **Other Undergraduate Students –** ECE 2400 can count as your second engineering distribution course. ECE 2400 is a great way to strengthen your programming skills and can nicely complement core classes in many other disciplines.

12. Policies

This section outlines various policies concerning auditors, usage of cellular phones and laptops in lecture, turning in assignments late, regrading assignments, collaboration, and accommodations for students with disabilities.

12.A Auditor Policy

Casual listeners that attend lecture but do not enroll as auditors are not allowed; you must enroll officially as an auditor. *If you would like to audit the course please talk to the instructor first!* Usually we wait until the second week of classes before allowing auditors to enroll, to ensure there is sufficient capacity in the lecture room. The requirements for auditors are: (1) attend most of the lectures; (2) take the short in-class quizzes; and (3) perform reasonably well on these quizzes. If you do not plan on attending the lectures for the entire semester, then please do not audit the course. Please note that students are not allowed to audit the course and then take it for credit in a later year unless there is some kind of truly exceptional circumstance.

12.B Course Re-Enrollment Policy

Students are not allowed to enroll for credit for a significant fraction of the course, drop or switch to auditor status, and then re-enroll for credit in a later year. A "significant fraction of the course" means after the first prelim; by this time the student will have: attended sevearl lectures, completed multiple programming assignments, and completed several short in-class quizzes. The student should have plenty of experience to decide whether or not they should drop and take the course in a later year. It is not fair for students to have access to assignment solutions and possibly even take the midterm before deciding to drop the course and take it again in a later year; this would essentially enable students to take the course twice to improve their grade.

12.C Cellular Phones and Laptops in Lecture Policy

Students are prohibited from using cellular phones and laptops in lecture unless they receive explicit permission from the instructor. It is not practical to take notes with a laptop for this course. Students will need to write on the handouts, quickly draw pipeline diagrams, and sketch microarchitectural block diagrams during lecture. The distraction caused by a few students using (or misusing) laptops during lecture far outweighs any benefit. Tablets are allowed as long as they are kept flat and used exclusively for note taking. If you feel that you have a strong case for using a laptop during lecture then please speak with the instructor.

12.D Late Assignment Policy

Programming assignment reports must be submitted electronically in PDF format and the code must be submitted electronically via GitHub. **No other formats will be accepted!** Programming assignments must be submitted by 11:59pm on the due date. No late submissions will be accepted and no extensions will be granted except for a family or medical emergency. We will be using the online CMS assignment submission system. You can continue to resubmit your files as many times as you would like up until the deadline, so please feel free to upload early and often. **If you submit an assignment even one minute past the deadline, then the assignment will be marked as late.**

As an exception to this rule, each student has two slip-days that may be used when submitting programming assignments throughout the semester. Each slip-day provides an automatic 24-hour extension. You may use up both slip-days on a single assignment, meaning that the maximum automatic extension is 48 hours. Students can pool their remaining slip days for the last two programming assignments. For example, if both students in a group have one slip day remaining before the fifth programming assignment, then they can use those two slip days on either the fifth or sixth programming assignment, or they can use one slip day for each. Regardless, the maximum automatic extension is 48 hours. To use a slip day, simply submit the report late; CMS will allow assignments to be uploaded up to two days late. You are responsible for keeping track of how many slip days you have remaining. If you accidentally submit an assignment late without the proper number of slip days remaining then, although the system will allow the upload, we will not grade the assignment (or we will grade the latest upload before the due date). The purpose of the slip-day system is to give you the freedom to more effectively manage your time. The due dates for the course are available at the beginning of the semester, so please plan ahead so you can handle weeks with many other deadlines.

12.E Regrade Policy

Addition errors in the total score are always applicable for regrades. Regrades concerning the actual solution should be rare and are only permitted when there is a significant error. Please only

make regrade requests when the case is strong and a significant number of points are at stake. Regrade requests should be submitted online via a private post on Piazza within one week of when an assignment is returned to the student. You must provide a justification for the regrade request.

12.F Collaboration Policy

The work you submit in this course is expected to be the result of your individual effort only, or in the case of the final two programming assignments, the result of you and your group's effort only. Your work should accurately demonstrate your understanding of the material. The use of a computer in no way modifies the standards of academic integrity expected under the University Code.

You are encouraged to study together and to discuss information and concepts covered in lecture with other students. You can give "consulting" help to or receive "consulting" help from other students. Students can also freely discuss basic computing skills or the course infrastructure. **However, this permissible cooperation should never involve one student (or group) having possession of or observing in detail a copy of all or part of work done by someone else, in the form of an email, an email attachment file, a flash drive, a hard copy, or on a computer screen.** Students are not allowed to seek consulting help from online forums outside of Cornell University. Students are not allowed to use online solutions (e.g., from Course Hero) from previous offerings of this course. Students are encouraged to seek consulting help from their peers and from the course staff via office hours and the online Piazza discussion forums. **If a student receives consulting help from anyone outside of the course staff, then the student must acknowledge this help on the submitted assignment.**

During examinations, you must do your own work. Talking or discussion is not permitted during the examinations, nor may you compare papers, copy from others, or collaborate in any way. Students must not discuss an exam's contents with other students who have not taken the exam. If prior to taking it, you are inadvertently exposed to material in an exam (by whatever means) you must immediately inform an instructor.

Should a violation of the code of academic integrity occur, then a primary hearing will be held. See `http://theuniversityfaculty.cornell.edu/dean/the-rules/academic-integrity1` for more information about academic integrity proceedings.

Examples of acceptable collaboration:

- Ben is struggling to complete a programming assignment which requires implementing an RPN calculator. He talks with Alice and Cathy and learn that all three students are really struggling. So the three students get together for a brainstorming session. They review the lecture and reading materials and then sketch on a whiteboard some ideas on how to implement an RPN calculator. They might also sketch out some code snippets to try and understand the best way to describe some of the software. Then each student independently writes the code for the assignment and includes an acknowledgment of the help they received from the other students. At no time do the students actually share code.
- Alice and Amy are having difficulty figuring out difficult test cases for their in-memory distributed cache. They post on Piazza to see if anyone has some general ideas for tricky corner cases. Ben and Bob figured out an interesting test case that ensures their in-memory distribute cache correctly handles replacing stale data, so Ben and Bob post a qualitative description of this test case. Alice and Amy independently write the code for this test case and then include an acknowledgment of the help they received from the other group. At no time do the groups actually share test code.

Examples of unacceptable collaboration:

- Ben is struggling to complete a programming assignment which requires implementing an RPN calculator. He talks with Alice and Cathy and learn that all three students are really struggling. So the three students get together for a joint coding session. Each student works on one part of the RPN calculator, then they combine these parts together to create the final working RPN calculator. *The three students share and copy each other's code often in order to finish the assignment.* Each student submits the final code independently. Each student acknowledges the help he or she received from the other students, but it doesn't matter since they explicitly shared code.
- Alice and Amy are having difficulty figuring out difficult test cases for their in-memory distributed cache. They post on Piazza to see if anyone has some general ideas for tricky corner cases. Ben and Bob figured out an interesting test case that ensures their in-memory distribute cache correctly handles replacing stale data, so *Alice and Amy send their test code to Ben and Bob via email.* Alice and Amy modify this test code and then include it in their submission. Alice and Amy include an acknowledgment of the help they received from the other group, but it doesn't matter since they explicitly shared code.

Notice that **the key is that students should not share the actual solutions or code with each other.** Consulting with your fellow students is fine and is an important part of succeeding in this course.

12.G Accommodations for Students with Disabilities

In compliance with the Cornell University policy and equal access laws, the instructor is available to discuss appropriate academic accommodations that may be required for students with disabilities. Requests for academic accommodations are to be made during the first three weeks of the semester, except for unusual circumstances, so arrangements can be made. Students are encouraged to register with Student Disability Services to verify their eligibility for appropriate accommodations.

13. Online and Computing Resources

We will be making use of a variety of online websites and computing resources.

- **Public Course Website –** `http://www.csl.cornell.edu/courses/ece2400`
  This is the main public course website which has the course details, updated schedule, reading assignments, and most handouts.
- **Piazza Discussion Forums –** `http://www.csl.cornell.edu/courses/ece2400/piazza`
  Piazza is an online question-and-answer platform. We will be using Piazza for all announcements and discussion on course content and programming assignments. We will enroll students that sign up for the course in Piazza. The course staff is notified whenever anyone posts on the forum and will respond quickly. Using the forum allows other students to contribute to the discussion and to see the answers. Use common sense when posting questions such that you do not reveal solutions. Please prefer posting to Piazza as opposed to directly emailing the course staff unless you need to discuss a personal issue.
- **CMS –** `http://www.csl.cornell.edu/courses/ece2400/cms`
  CMS is an online assignment management system developed by the Cornell Computer Science department. Students will use CMS for submitting the programming assignment reports. We will enroll students that sign up for the course in CMS. We will also be posting restricted materials (e.g., quizzes, solutions) on the CMS site, and we will use CMS to distribute grades. We will not be using CMS to post announcements.

- **ECE Computing Resources –**
  The ECE department has a cluster of Linux-based workstations and servers which we will be using for the programming assignments. You can access the ECE computing resources by using the ECE Linux Computing Lab in 314 Phillips Hall, you can use the CIT Windows Computing Lab in 318 Phillips Hall, or you can log into the `ecelinux` servers remotely from your own personal workstation. You do not need a special account; you will instead simply use your NetID and Cornell password to log into the ECE computing resources.
- **GitHub –** `http://www.csl.cornell.edu/courses/ece2400/github`
  GitHub is an online Git repository hosting service. We will be using GitHub to distribute programming assignment harnesses and as a mechanism for student collaboration on the final two programming assignments. Students will also use GitHub for submitting the code for their programming assignments. Students are expected to become familiar with the Git version control system.
- **TravisCI –** `http://www.csl.cornell.edu/courses/ece2400/travisci`
  TravisCI is an online continuous integration service that is tightly coupled to GitHub. TravisCI will automatically run all tests for a students' programming assignment every time the students push their code to GitHub. We will be using the results reported by TravisCI to evaluate the code functionality of the programming assignments.