1. Course Information

Cross-Listings
ENGRD 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
Jennifer Sandora, 314 Rhodes Hall, jes633@cornell.edu

Graduate TAs
Tuan Ta qtt2 Office/Lab Hours: 314 Phillips, Wed, 7:30–9:30pm
Hongyi Deng hd294 Office/Lab Hours: 314 Phillips, Mon, 7:30–9:30pm

Undergraduate TAs
Yixiao Zhang yz624 Office/Lab Hours: 314 Phillips, Tue, 7:30–9:30pm
Nicole Kwok njk72 Office/Lab Hours: 314 Phillips, Mon, 7:30–9:30pm
Ho-Jung Yang hj379 Office/Lab Hours: 314 Phillips, Wed, 7:30–9:30pm
Fareeza Hasan fh244 Office/Lab Hours: 314 Phillips, Tue, 7:30–9:30pm

Lectures
203 Phillips Hall, Mon/Wed/Fri, 10:10–11:00am

Disc. Section
403 Phillips Hall OR 314 Phillips Hall, Friday, 2:30–4:25pm

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
dce2400-staff-1@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 the principles, practices, and art of computer systems programming using C and C++, the languages of choice for system-level programmers.

The course lectures are structured into three parts. In the first part, students will use C to explore procedural programming (e.g., functions, conditional statements, and iteration statements; recursion; types; pointers; arrays; dynamic allocation); students will then apply their knowledge of C to explore basic data structures and algorithms (e.g., complexity analysis; concrete data types such as lists and vectors; abstract data types such as sequences, stacks, queues, sets, and maps; sorting
algorithms). In the second part, students will transition from C to C++ and then use C++ to explore four programming paradigms: object-oriented programming (e.g., C++ classes and inheritance for dynamic polymorphism), generic programming (e.g., C++ templates for static polymorphism), functional programming (e.g., C++ functors and lambdas), and concurrent programming (e.g., C++ threads and atomics); students will then apply their knowledge of multi-paradigm C++ to explore more advanced data structures and algorithms involving trees, tables, and graphs. In the third part, students will explore systems programming in the UNIX environment using the POSIX standard library.

The course includes a series of programming assignments for students to put the principles they have learned into practice. Students will gain experience with UNIX software development including command line development tools, distributed version control, unit testing frameworks, continuous integration, debugging tools, and performance evaluation. In the final programming assignment, students will work in pairs to design, implement, test, and evaluate a high-performance handwriting recognition system which uses machine learning to automatically classify handwritten letters.

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:

- **describe** both basic and advanced algorithms and data structures and how to analyze these algorithms and data structures in terms of time and space complexity;
- **apply** the C and C++ programming languages to implement algorithms and data structures using a variety of programming paradigms;
- **evaluate** various algorithm and data structure alternatives and make a compelling theoretical and/or practical argument for why one approach is superior;
- **create** non-trivial C/C++ programs (roughly 1,000 lines of code) and the associated testing strategy starting from an English language specification; and
- **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 Procedural Programming, Basic Data Structures & Algorithms** – variables, expressions, functions, conditional statements, iteration statements, recursion, static types, pointers, arrays, dynamic allocation, lists, vectors, complexity analysis, abstract data types, sequences, stacks, queues, sets, maps, sorting algorithms

- **Part 2: C++ Multi-Paradigm Programming, Advanced Data Structures & Algorithms** – transition to C++, namespaces, flexible function prototypes, references, exceptions, new/delete, object oriented programming (e.g., C++ classes and inheritance for dynamic polymorphism), generic programming (e.g., C++ templates for static polymorphism), functional programming (e.g., C++ functors and lambdas), concurrent programming (e.g., C++ threads and atomics), trees, tables, graphs

- **Part 3: Systems Programming in the UNIX Environment** – POSIX standard library including I/O and processes

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 includes 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 several additional books that students may find useful for providing additional background or more advanced material. Many of these books are on reserve at Uris Library and/or are available as ebooks through the Cornell library.

8. Format and Procedures

This course includes a combination of lectures, short quizzes, discussion sections, piazza participation, assigned readings, programming assignments, and exams.

- **Lectures** – Lectures will be from 10:10am to 11:00am every Monday, Wednesday, and Thursday in 203 Phillips Hall excluding the following academic holidays: Labor Day (9/3), Indigenous People’s Day (10/8), and Thanksgiving (11/21,11/23). 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 due at the beginning of some lectures. Some quizzes will be completed in lecture while others should be completed before lecture. Regardless, quizzes should always be completed independently with absolutely no collaboration (either with students or instructors). Take-home quizzes are due promptly at the beginning of lecture. Students must turn in their take-home quiz in person. Students cannot give their take-home quiz to another student to turn in. The quiz should take about 5–10 minutes, and will cover some of the key topics discussed in the previous lecture. There are no make-up quizzes. The lowest quiz score is dropped which effectively provides for one excused absence. Solutions to quizzes will be available online soon after the quiz is given for formative self-assessment.

- **Discussion Section** – The discussion section will be on Fridays from 2:30pm to 4:25pm in either 403 Phillips Hall or 314 Phillips Hall. Attendance at the weekly discussion sections is required. Note that we will often divide the two-hour section into two one-hour sections; in these cases, students are only required to attend one of these one-hour sections. 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.

- **Online Discussion** – Engagement via the Piazza online discussion forums is a critical component of the course. Students are required to actively participate on Piazza, and this participation will be assessed as part of each student’s final grade. There are many ways student can participate including posting questions, liking other questions, responding that they have the same question, helping to create answers, and posting interesting related content.

- **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 five programming assignments. Students are expected to work individually on the first four programming assignments and in a group on the final programming assignment. Students will be using the ECE Computing Resources to complete the programming assignments and the code and report must be submitted via GitHub (see Section 13). The report must be submitted in PDF format. No other means of submission will be accepted. Programming assignments are due on Thursdays at 11:59pm except for the final programming assignment which is due on a Tuesday at 11:59pm (see Section 12.D for late assignment policy).

- **Prelim and Final Exams** – The course includes two prelim exams and a cumulative final exam. The exams assess student understanding of the material presented in lecture and assigned read-
ings. The exams do not assess student understanding of the Linux development environment (e.g., the Linux command line, make, gdb, etc.) used for the programming assignments. 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 Thursdays at 11:59pm except for the final programming assignment which is due on a Tuesday at 11:59pm. Changes to this schedule will be posted as announcements via Piazza.

Thu Sep 13 PA1 – Incremental Milestone
Thu Sep 20 PA1 – Math Functions
Thu Sep 27 PA2 – Incremental Milestone
Thu Oct 4 PA2 – List and Vector Data Structures
Thu Oct 11 Prelim #1 from 7:30-9:30pm in 219 Phillips
Thu Oct 18 PA3 – Incremental Milestone
Thu Oct 25 PA3 – Sorting Algorithms
Thu Nov 1 PA4 – Incremental Milestone
Thu Nov 8 PA4 – Polymorphic Data Structures and Algorithms
Thu Nov 15 Prelim #2 from 7:30-9:30pm in 110 Hollister
Thu Nov 22 Thanksgiving Break
Tue Nov 27 PA5 – Incremental Milestone
Tue Dec 4 PA5 – Handwriting Recognition System
Sat Dec 8 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 will be 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. Piazza points are based on active participation the Piazza online discussion forums. The weighting for the various assignments is shown below.

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Piazza Points</td>
<td>2%</td>
</tr>
<tr>
<td>Quizzes</td>
<td>8%</td>
</tr>
<tr>
<td>PA Milestones</td>
<td>5%</td>
</tr>
<tr>
<td>PA Code</td>
<td>20%</td>
</tr>
<tr>
<td>PA Reports</td>
<td>10%</td>
</tr>
<tr>
<td>Prelim Exams</td>
<td>30%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>25%</td>
</tr>
</tbody>
</table>

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 four out of the five 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 / ENGRD 2140 can be used as your second engineering distribution course (all ECE students must take ECE 2300 as their first engineering distribution course). Alternatively, ECE students may be able to use this course as an outside-ECE technical elective. Students are reminded they are only allowed to use one ECE course as an outside technical elective provided that the course’s subject matter lies outside the student’s major disciplinary area as determined by the focus of the student’s upper-level ECE course work. Regardless, this course satisfies the ECE advanced programming requirement.

- **ECE M.Eng. Students** – ECE 2400 can be used as a technical elective course, although students are reminded they can only use one ECE undergraduate course as a technical elective.

- **CS Undergraduate Students** – All CS majors must take CS 2110 / ENGRD 2110 as their first engineering distribution course. Since both CS 2210 / ENGRD 2110 and ECE 2400 / ENGRD 2140 are part of the same “scientific computing” ENGRD category, you cannot use ECE 2400 as your second engineering distribution course. However, CS majors can still use ECE 2400 as an advisor-approved elective or a major-approved free elective.
Other Undergraduate Students in the College of Engineering – Students are required to take two ENGRD courses from different categories. ECE 2400 / ENGRD 2140 is part of the “scientific computing” category, so students can use this course as their second engineering distribution course assuming the first is not also in the “scientific computing” category (i.e., not ENGRD 2110, ENGRD 2112, ENGRD 3200). ECE 2400 / ENGRD 2140 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, copyright, 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) complete the short 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 several lectures, completed multiple programming assignments, and completed several 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 both prelims 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 state diagrams, and sketch pseudocode 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 in PDF format and both the code and the report must be submitted electronically via GitHub. No other means of submission will be accepted! Program-
ECE 2400 Computer Systems Programming, Fall 2018
Course Syllabus

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 an online 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 five slip-days that may be used when submitting programming assignments (both incremental milestones and the final assignment) 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. 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. Re-grade 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 programming assignment, 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 take-home quizzes, in-class quizzes, and 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 a quiz/exam’s contents with other students who have not taken the quiz/exam. If prior to taking it, you are inadvertently exposed to material in an quiz/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 https://theuniversityfaculty.cornell.edu/academic-integrity for more information about academic integrity proceedings.

Examples of acceptable collaboration:

- Bob is struggling on a take-home quiz about object oriented programming, so he seeks consulting help from Alice, a fellow student in the course. Alice goes through various examples from the lecture and reading materials to help Bob understand the concepts, and they sketch a few state diagrams related to different problems (not the quiz problem) on a whiteboard. Bob and Alice work independently on the take-home quiz. At no time do Bob and Alice actually collaborate on the take-home quiz so there is no need for Bob to acknowledge the consulting help he received from Alice.

- Ben is struggling to complete a programming assignment which requires implementing a linked list. He talks with Alice and Cathy and learns 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 a linked list. They might also sketch out some code snippets to try and understand the best way to implement the data structure. 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 handwriting recognition system. 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 handwriting recognition system correctly handles the worst case input, 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:

- Bob is struggling on a take-home quiz about object oriented programming, so he seeks consulting help from Alice, a fellow student in the course. Alice shows Bob her completed quiz solution and walks him through the various steps required to solve the quiz. Bob takes some notes during their discussion, and then independently writes up his solutions. Bob acknowledges the help he received from Alice on his submission, but it doesn’t matter since Alice explicitly shared her solution with Bob.

- Anna has finished the take-home quiz, but she cannot make today’s lecture since she has an extracurricular commitment. Bart has also finished the quiz and is planning to attend today’s lecture. Anna gives her quiz to Bart, and Bart turns in both quizzes together. This misrepresents Anna’s attendance and is not allowed.

- Ben is struggling to complete a programming assignment which requires implementing a linked list. He talks with Alice and Cathy and learns that all three students are really struggling. So the three students get together for a joint coding session. Each student works on one method of the linked list class, and then they combine these methods together to create the final working linked list class. 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 handwriting recognition system. 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 handwriting recognition system correctly handles the worst case input, 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 Copyright Policy

All course materials produced by the course instructor (including all handouts, tutorials, homeworks, quizzes, exams, videos, scripts, and code) are copyright of the course instructor unless otherwise noted. Download and use of these materials are permitted for individual educational non-commercial purposes only. Redistribution either in part or in whole via both commercial (e.g., Course Hero) or non-commercial (e.g., public website) requires written permission of the copyright holder.

12.H 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. Please post non-anonymously if at all possible.

• ECE Computing Resources – ecelinux.ece.cornell.edu
  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 the commercial GitHub service 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. Note that we are not using the Cornell hosted version of GitHub as in some other courses; we are using github.com.

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 help evaluate the code functionality of the programming assignments.

Codecov.io – http://www.csl.cornell.edu/courses/ece2400/codecovio
Codecov.io is an online code-coverage visualization service that is tightly coupled to GitHub and TravisCI. Codecov.io will automatically display code-coverage reports after each TravisCI build. We will be using the results reported by Codecov.io to help evaluate the verification quality of the programming assignments.