15-122: Principles of Imperative Computation

Course Syllabus
Spring 2019
Saquib Razak

January 14, 2019

This course teaches imperative programming and methods for ensuring the correctness of programs. It is intended for students with a basic understanding of programming (variables, expressions, loops, arrays, functions). Students will learn the process and concepts needed to go from high-level descriptions of algorithms to correct imperative implementations, with specific application to basic data structures and algorithms. Much of the course will be conducted in a subset of C amenable to verification, with a transition to full C near the end.

Either 21-127 *Concepts of Mathematics* or 15-151 *Mathematical Foundations for Computer Science* is a co-requisite (must be taken before or in the same semester). This course is a prerequisite for 15-213 *Computer Systems* and 15-210 *Parallel and Sequential Data Structures and Algorithms*.

1 Course Objectives

We categorize learning outcomes into *computational thinking*, *programming skills* and *data structures and algorithms*.

Computational Thinking: Students should leave this course able to explain abstraction and other key computer science concepts, apply these fundamental concepts as problem-solving tools, and wield *contracts* as a tool for reasoning about the safety and correctness of programs.[1] In particular, we expect students to be able to:

[1] The importance of *computational thinking* as a fundamental skill has been discussed by Jeannette Wing (http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/Wing06.pdf) and by the Computer Science Teachers Association (http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf).

1. Develop contracts (preconditions, postconditions, assertions, and loop invariants) that establish the safety and correctness of imperative programs.
2. Develop and evaluate proofs of the safety and correctness of code with contracts.
3. Develop and evaluate informal termination arguments for programs with loops and recursion.
4. Evaluate claims of both asymptotic complexity and practical efficiency of programs by running tests on different problem sizes.
5. Define the concept of programs as data, and write programs that use the concept.
6. Defend the use of abstractions and interfaces in the presentation of algorithms and data structures.
7. Identify the difference between *specification* and *implementation*.
8. Compare different implementations of a given specification and different specifications that can be applied to a single implementation.
9. Explain data structure manipulations using data structure invariants.
10. Identify and evaluate the use of fundamental concepts in computer science as problem-solving tools:
    (a) Order (sorted or indexed data),
    (b) Asymptotic worst case, average case, and amortized analysis,
    (c) Randomness and (pseudo-)random number generation, and
    (d) Divide-and-conquer strategies.

Programming Skills: Students should leave this course able to read and write code for imperative algorithms and data structures. In particular, we expect students to be able to:

1. Trace the operational behavior of small imperative programs.
2. Identify, describe, and effectively use basic features of C0 and C:
    (a) Integers as signed modular arithmetic,

(b) Integers as fixed-length bit vectors,
(c) Characters and strings,
(d) Boolean operations with short-circuiting evaluation
(e) Arrays,
(f) Loops (`while` and `for`),
(g) Pointers,
(h) Structs,
(i) Recursive and mutually recursive functions,
(j) Void pointers and casts between pointer types,
(k) Contracts (in C0)
(l) Casts between different numeric types (in C),

3. Translate between high-level algorithms and correct imperative code.
4. Translate between high-level loop invariants and data structure invariants and correct contracts.
5. Write code using external libraries when given a library interface.
6. Develop, test, rewrite, and refine code that meets a given specification or interface.
7. Develop and refine small interfaces.
8. Document code with comments and contracts.
9. Identify undefined and implementation-defined behaviors in C.
10. Write, compile, and test C programs in a Unix-based environment using `make`, `gcc`, and `valgrind`.

Algorithms and Data Structures: Students should leave this course able to describe the implementation of a number of basic algorithms and data structures, effectively employ those algorithms and data structures, and explain and interpret worst-case asymptotic complexity arguments. In particular, we expect students to be able to:

1. Define and describe big-O notation, both formally and informally.
2. Compare common complexity classes like $O(1)$, $O(n)$, $O(n * \log(n))$, $O(n^2)$, and $O(2^n)$.

3. Explain the structure of basic amortized analysis proofs that use potential functions.
4. Apply principles of asymptotic analysis and amortized analysis to new algorithms and data structures.
5. Recognize properties of simple self-adjusting data structures.
6. Recognize algorithms and data structures using divide-and-conquer.
7. Describe and employ a number of basic algorithms and data structures:
   (a) Integer algorithms,
   (b) Linear search,
   (c) Binary search,
   (d) Sub-quadratic complexity sorting (mergesort and quicksort),
   (e) Stacks and queues,
   (f) Pseudo-random number generators,
   (g) Hash tables,
   (h) Priority queues,
   (i) Balanced binary search trees,
   (j) Disjoint-set data structures (union/find), and
   (k) Simple graph algorithms.

2 Resources

Lecture notes: There is no textbook for this course; lecture notes and other resources will be provided through the "Schedule" page of the course website.

Online resources: All course resources will all be linked from the course home page.

- Home page (https://www.qatar.cmu.edu/~srazak/courses/15122-s19)
  Schedule, office hours, lecture notes, written homework hand-in.
- Autolab (https://autolab.andrew.cmu.edu)
  Grades, programming homework hand-in, style comments.

- Piazza (https://piazza.com)

  Announcements, discussions, questions.

Archived exams: Midterms and final exams from previous semesters, as well as sample solutions, will be posted before each exam.

Programming tools: In the first nine weeks the course uses C0, a small safe subset of C augmented with a layer to express *contracts*. A tutorial for this language is available online at http://c0.typesafety.net/tutorial/. This language has been specifically designed to support the student learning objectives in this course. In particular it provides garbage collection (freeing students from dealing with low-level details of explicit memory management), fixed range modular integer arithmetic (avoiding complexities of floating point arithmetic and multiple data sizes), an unambiguous language definition (guarding against relying on undefined behavior), and contracts (making code expectations explicit and localize reasoning).

In the last six weeks the course transitions to C, in preparation for subsequent systems courses. Emphasis is on transferring positive habits developed in the use of C0, and on practical advice for avoiding the pitfalls and understanding the idiosyncrasies of C. We use the `valgrind` tool to test proper memory management.

3 Student Evaluation

The course is graded on a 1000-point scale.

- Homework (450 points)
  - Weekly programming assignments (usually due Thursday 10pm)
  - Weekly written assignments (usually due Monday 5pm)
- 1 Final (250 points), date TBA
- 2 Midterms (125 points each), Check course website for exact dates
- In-Lecture/In-Recitaiton Quizzes and Labs (50 points max)

Labs and quizzes are primarily intended to help you and are graded on a "check minus, check, check plus" scale. On a point scale, this is worth 1, 2, and 3 points, respectively. The 50 points available for lab credit can be obtained with a "check" for every lab and quiz. The maximum combined quiz and lab grade is 50, even though it is possible to get significantly more than 50 points.

3.1 Style grading

Students are expected to write code with good programming style; a review of what constitutes good style is given at `http://www.cs.cmu.edu/~rjsimmon/15122-s15/etc/styleguide.pdf`.

For a small subset of assignments (likely 2 or 3), TAs will review all final submissions by hand. If there are significant style issues, they may give a non-passing grade on style, accompanied by "FIX STYLE" annotations in their code. Students who are told to redo their style *must* address these issues and discuss their revisions with a TA within **ten days** of the style grades being posted. Any TA or instructor can do style re-grading at any office hour; you *do not* have to go to the TA that assigned the grade.

3.2 Grade appeals

After each exam and homework assignment is graded, your score will be posted on the Autolab gradebook. We will make the utmost effort to be fair and consistent in our grading. Any TA is permitted to fix simple arithmetic errors (and, at their discretion, other blindingly obvious grading errors). For any other grading issues, you *must* request a regrade as follows:

- Write or print a **hardcopy** letter explaining in detail where and why you think there was a mistake in grading.
- Hand-deliver the cover letter to Saquib Razak. Slide them under his door if he is not in. **Verbal or email requests will NOT be accepted.**
- Every written homework and midterm exam is handed back in a Monday lab. All regrade requests must be recieved within **ten days** of the work being handed back in lab or (for all graded work not handed back in lab) within ten days of the grade being posted to Autolab.

(This policy adapted from 15-213. Thanks, 15-213!)

3.3 Final grades

Absent exceptional circumstances, students with a total score of 900 and above will get an A, 800 and above will get a B, 700 and above a C, and 600 and above a D. This assignment assumes that the makeup of a student's grade is not wildly anamalous: exceptionally low overall scores on exams, programming assignments, or written assignments may be treated as an exceptional circumstance.[2]

Grade cutoffs may be lowered based on the difficulty of exams and assignments. Precise grade cutoffs will not be discussed at any point during or after the semester.

For students very close to grade boundaries, instructors may, at their discretion, consider participation in lecture and recitation and exam performance when assigning the final grade.

4 Policies

4.1 Class presence and participation

Active participation by you and other students will ensure that everyone has the best learning experience in this class. We may take participation in lecture and recitation (the one you are *registered* for) into account when setting final grades. Therefore, please attend the lecture, lab, and recitation you are registered for.

[2] Almost a quarter of the students who recieved a B in Fall 2014 had a 90%-100% average on programming assignments, an 80%-90% average on written homeworks, and a 70%-80% average on exams. Grades distributed along these lines will therefore, of course, not be treated as anamalous.

4.2 Laptops and mobile devices

As research on learning shows, unexpected noises and movement automatically divert and capture people's attention, which means you are affecting everyone's learning experience if your cell phone, pager, laptop, etc. makes noise or is visually distracting during class.

Therefore, please silence all mobile devices during class. Do not work on assignments for this or any other class while attending lecture or recitation.

4.3 Deadlines

There will generally be two deadlines every week (the days will change during the second half of the semester. Refer to the course website and homework handouts for accurate dealines:

- 5:00pm Monday - Written homework due on gradescope.
- 10pm Thursday - Programming homework due (via Autolab).

4.4 Late work

- *Programming*: each student has four grace days, *at most one of which* can be used on any of the 11 programming assignments to extend the deadline by 24 hours to 10pm on Friday (assuming a Thursday 10pm deadline). Late submissions from students who have exhausted their late days will receive no credit, and no work will be accepted more than 24 hours after the original deadline.
- *Written*: Grace days cannot be used for written homeworks.

4.5 Personal accommodations

Reasonable exceptions to the late work policy may be made for family, religious, athletic, or other external obligations *if* we know about them well in advance (at least one week, assuming circumstances make this possible). If you have other extenuating circumstances, such as an extended illness that affects your ability to complete the course, please notify the instructors.

We do not do makeups for quizzes. If you will miss a significant number of labs, recitations, or lectures for a valid reason, we will take this into account when assigning final grades.

Students with disabilities: If you wish to request an accommodation due to a documented disability, please inform your instructor and contact Disability Resources as soon as possible . Special accommodation for exams will be coordinated by instructor, and must be requested for *each* exam separately a week in advance. Other requests for deadline flexibility will be handled on a case-by-case basis.

4.6 Academic integrity

The university policies and procedures on academic integrity will be applied rigorously.

The value of your degree depends on the academic integrity of yourself and your peers in each of your classes. It is expected that, unless otherwise instructed, the work you submit as your own will be your own work and not someone else's work or a collaboration between yourself and other(s).

Please read the University Policy on Academic Integrity carefully to understand the penalties associated with academic dishonesty at Carnegie Mellon. In this class, cheating/copying/plagiarism means copying all or part of a program or homework solution from another student or unauthorized source such as the Internet, knowingly giving such information to another student, or giving or receiving unauthorized information during an examination. In general, *each solution you submit (quiz, written assignment, programming assignment, midterm or final exam) must be your own work*. In the event that you use information written by another person in your solution, you must cite the source of this information (and receive prior permission if unsure whether this is permitted). It is considered cheating to compare complete or partial answers, discuss details of solutions, or sit near another person who is taking the same course and try to complete the assignment together.

Your course instructor reserves the right to determine an appropriate penalty based on the violation of academic dishonesty that occurs. *Violations of the university policy are likely to result in severe penalties including failing this course and possible expulsion from Carnegie Mellon University.* If you have any questions about this policy and any work you are doing in the course, please feel free to contact your instructor for help.

We will be using the Moss system to detect software plagiarism.

It is not considered cheating to clarify vague points in the assignments, lectures, lecture notes, or to give help or receive help in using the computer systems, compilers, debuggers, profilers, or other facilities. It is not cheating to review graded assignments or exams with students in the same class

as you, but it *is* considered unauthorized assistance to share these materials between different iterations of the course.