15-122: Principles of Imperative Computation

Course Syllabus
Frank Pfenning

August 28, 2011

This course is intended for students with a basic understanding of programming (variables, expressions, loops, arrays, functions). It teaches imperative programming and methods for ensuring the correctness of programs. 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. 21-127 *Concepts of Mathematics* is a co-requisite (must be taken before or in the same semester). This course is a pre-requisite for 15-213 *Computer Systems* and 15-210 *Parallel and Sequential Data Structures and Algorithms*.

1 Learning Outcomes

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

Computational Thinking

In the area of *computational thinking*, students will be able to

- Understand abstractions and interfaces.
- Relate specifications to implementations.
- Express pre- and post-conditions for functions and loop invariants.
- Use data structure invariants.
- Reason rigorously about code, both logically and operationally.

- Analyze asymptotic complexity and practical efficiency.
- View programs as data.

Programming Skills

In the area of *programming skills*, students will be able to

- Understand the static and dynamic semantics of their programs.
- Develop, test, rewrite, and refine their code.
- Work with specifications and invariants.
- Use and design small API's.
- Use and implement mutable data structures.
- Render high-level algorithms into correct imperative code.
- Write C programs in a Unix-based environment.

Data Structures and Algorithms

In the area of *data structures and algorithms*, students will be able to

- Perform asymptotic analysis on sequential computation, including simple amortized analysis and recognition of common complexity classes ($O(n), O(n * \log(n)), O(n^2), O(2^n)$).
- Apply the divide-and-conquer strategy in algorithm design.
- Understand properties of simple self-adjusting data structures.
- Effectively employ a number of basic algorithms and data structures, including binary search, subquadratic sorting, stacks and queues, hash tables, priority queues, balanced binary search trees, tries, binary decision diagrams, simple graph algorithms.

2 Programming Language

In weeks 1–11 the course uses C0, a small safe subset of C augmented with a layer texpression *contracts*. 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 weeks 11–15 the course transitions to C, in preparation for subsequent systems courses. Emphasis is on tranferring 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 for proper memory management.

3 Course Materials

There is at present no textbook for this course, but detailed lecture notes as well as a programming language reference. Course materials can be found at:

- Fall 2010: `http://www.cs.cmu.edu/~fp/courses/15122-f10`
- Spring 2011: `http://www.cs.cmu.edu/~fp/courses/15122-s11`
- Current: `http://www.andrew.cmu.edu/course/15-122`

4 Student Evaluation

Students will be evaluated based on the following components:

- 8 Assignments (total 45%), each with a written and a programming component.
- 1 Final (total 25%) of 180 minutes.
- 2 Midterms (total 20%) of 80 minutes each.
- 8 Quizzes (total 10%), taken on-line.

A total course score of 90% and above is guaranteed an A, 80%-90% a B, etc. but grade cutoffs may be lowered based on the difficulty of exams and assignments. For students near grade boundaries, class participation and extra credit will be considered at the instructors discretion.

5 Schedule

Course schedule will vary somewhat from semester to semester; the following is a sample schedule.

- Week 1: Course overview, contracts, introduction to C0 (functions, statements, expressions, types)
- Week 2: Modular arithmetic, arithmetic and bitwise operations, arrays, loop invariants.
- Week 3: Linear and binary search, divide and conquer, asymptotic complexity.
- Week 4: Sorting algorithms, mergesort, quicksort.
- Week 5: Queues, stacks, linked lists, pointers, recursive types, data structure invariants.
- Week 6: Memory layout, recursion, *Midterm Exam I*.
- Week 7: Unbounded arrays, amortized analysis, hash tables.
- Week 8: Interfaces, priority queues, heaps, ordering and shape invariants.
- Week 9: Restoring invariants, heapsort, binary search trees.
- Week 10: AVL trees, rotations, program testing.
- Week 11: Polymorphism, introduction to C, *Midterm Exam II*.
- Week 12: Memory management (malloc/free), `valgrind`, generic data structures.
- Week 13: Virtual machines.
- Week 14: Tries, decision trees, binary decision diagrams, sharing, canonicity.
- Week 15: Graph algorithms (spanning trees, union-find).