Harvey Mudd College
Syllabus, Fall 1992

Computer Science 60
Principles of Computer Science

Instructor Robert M. Keller, email:keller@jarthur, tel. 621-8483, secretary: Nancy, 621-8225

Catalog Description

Introduction to principles of computer science. Algorithms, complexity analysis, data structuring, data and procedural abstraction, grammars, correctness, logic principles, processor organization, operating system concepts, programming languages, basic automata theory, and theoretical limitations.

Course Goals

To learn and apply basic principles of computer science, including software construction, hardware organization, and limitations of computers.

Texts (initials are used for the reading material in the course outline):

AU Alfred V. Aho and Jeffrey D. Ullman, *Foundations of computer science*. W.H. Freeman and Company, 1992. ISBN 0-7167-8233-2. This provides material on algorithms, data structures, analysis, theory, and various system aspects.

AT Andrew S. Tanenbaum, *Structured computer organization*. Prentice-Hall, 1990. ISBN 0-13-854662-2 This provides an introduction to computer architecture. It also touches upon system software, thus providing a bridge to the material in **AU**.

LA Leendert Ammeraal, *C++ for programmers*. Wiley (1991). ISBN 0-471-93011-3. Provides an introduction to the philosophy and usage of the C++ language.

Assignments and Grading

There will be about 10 assignments entailing programming, to help internalize key working concepts. The language used will mostly be C++, due to its position as an emergent industry standard and its ability to reflect both low-level computer structure and at the same time provide high-level data abstractions. Prior experience in C or C++ is not expected, however you should have moderate proficiency in Pascal programming. Although the course entails programming, we do not consider it to be primarily a course *in* programming.

The programming assignments will constitute 60% of the grade. The mid-term examination will count 15% and the final 25%. In addition, written "practice problems" will be indicated. These do not need to be submitted, but it will be a good idea to know how to work them for the exams. Exams are open book and emphasize conceptual understanding, not memorization of fine details.

Late Assignment Policy

Submissions are done by emailing to the grader, cs60grad@jarthur, which also establishes the time of submission. Turn in assignments whenever you wish. However, to get *full credit* on an assignment, turn it in by the *due date*. Here is the formula for discounting late homework: Any time before midnight on the due date is on time. Midnight or after is counted as the following day. If your homework is N days late, then your grade is decreased by 10N% of its otherwise full value. (Of many techniques with which we have experimented, this seems like a fair way to emphasize the need to generally get work done in a timely fashion, give flexibility on due times, and not penalize the student who does meet deadlines.)

Resources

We want the course to be interactive, and are eager to prevent programming difficulties from consuming extraordinary time, so please *ask* when you get stuck with such difficulties. It is much more efficient to start early on each assignment so that you give yourself enough time. You can get help on-line by mailing to the instructor (keller@jarthur) or the grader (cs60grad@jarthur). (Please do not use 'talk' to try to get help– it is disruptive). You are welcome to submit email or a note card with any question you'd like to have answered or any point you'd like addressed before or after class, or leave it in my mailbox in 111 Jacobs. Of course you may ask such questions orally as well. There are many tools available on the computer itself. Use the 'man' feature of Unix to find what you need and to explore. Helpful information, examples, on-line copies of assignments, etc. will be kept in subdirectories of the directory /cs/cs60.

CS 60 Topic Outline

The lectures will roughly follow this outline. The progression is at the rate of about two of the numbered topics below per week. Please keep up on the reading without it being explicitly assigned. The lectures and readings are intended to be generally complementary; sometimes they overlap but usually one augments the other. More often than not, several threads will be interwoven in the lectures over a period of time, in part to emphasize the commonality of concepts from different vantage points. We skip around in the texts quite a bit. They should be treated somewhat like handbooks, reading for conceptual ideas then referring back for details. Additional material will be provided in the form of handouts, typically for those topics in which there is no text material.

1. Computation problems and models of computation. States and transitions. Turing machines. Literate programming. Von Neumann computer (RAM) model. Functional specifications. Partial recursive functions. Cellular automata. Spreadsheets. Processes.
2. Propositional and predicate logic [AU sec. 12.1-12.5, 12.7-12.10, 14.1-14.4]. Programming in logic.
3. Encoding data. Number representations and conversions [AU sec. 4.11-4.12, AT Appendix A, B]. Lists. Streams.
4. Directed graphs [AU sec. 9.1-9.3]. Linked structures [AU sec. 6.1-6.5, 5.2-5.3].
5. Notions of abstraction. Data abstraction [AU ch. 1]. Stacks [AU sec. 6.6]. Queues [AU sec. 6.8]. Classes and objects [LA pages 1-105]. Priority queues [AU sec. 5.10], event-driven simulation.
6. Memory organization, linear-addressing, dynamic storage allocation [AU sec. 4.4-4.5, LA 125-148]. Pointer arithmetic [LA pages 106-124]. L-values and R-values.
7. Inductive definitions, datatype equations. Trees [AU sec. 5.6]. S-expressions. Grammars [AU sec. 11.2-11.6, 11.8] and abstract syntax. L-systems.
8. Recursion, divide-and-conquer [AU sec. 2.6-2.8, 2.2-2.4, 3.10, 5.4]. Transformation from iteration to recursion. Caching, dynamic programming [AU 6.9].
9. Program specification. Exceptions. Correctness of programs [AU sec. 2.5, 2.9]. Partial vs. total correctness. Termination. Analysis by compression and weakest preconditions.
10. Analyzing program performance empirically and mathematically [AU ch 3]. O notation. Growth-rate comparisons. Upper bounds.
11. Set and dictionary (mapping) ADTs [AU ch. 7], inverting mappings, binary search [AU pp 290-291]. Binary search trees [AU sec. 5.8], balanced trees. Hashing [AU sec. 7.6].
12. Examples of analysis from sorting [AU various places, sec. 3.9, 5.11]. Lower bounds.
13. Exam covering material thus far. **Date of examination: 2:45-4:00pm, 7 October 1992**
14. Customization and inheritance among classes [LA pages 149-168, 187-198]
15. Searching and path-finding problems in graphs [AU sec. 9.6-9.9]. Floyd/Warshall/Dijkstra algorithms. Traveling salesman problem. Backtracking.
16. Regular expressions and text matching [AU ch 10].
17. Floating point and numeric programming.
18. Logic gates and Boolean algebra. MSI components [AU ch 13, AT sec. 4.1]. Coding and combinatorial logic design. Karnaugh maps [AU 12.6]. Hypercubes. Iterated consensus.
19. Sequential logic design. Sequential equations. Latches, flip-flops. Moore vs. Mealy machines. The glitch phenomenon.
20. Processing unit structure [AU sec. 4.1-4.3, 4.6-4.8, AT ch. 2, 3, sec. 4.2-4.6]. Registers, buses, ALUs.
21. Instruction sets and addressing modes [AT ch. 5]. Assemblers [AT ch. 7].
22. Memory structure, virtual memory, operating systems [AT ch. 6].
23. Parallel computing, communication, computer networks.
24. Limitations: Universal machines; Incomputability. The busy beaver. Intractability and NP-completeness. Problem reduction [AU sec. 14.10].
25. Comprehensive final exam. **Date of examination: 2-5pm, 16 December 1992.**

Reading List

__ Turing Machines handout, read in conjunction with
__ Mathematical Basics handout
__ AU sec.12.1-12.4 (propositional logic)
__ AU 646-647 (substitution principle)
__ AU sec. 14.1-14.6 (predicate logic)
__ AU sec. 4.11-4.12 (number representation)
__ AT Appendix A, B (number representation)
__ AU sec. 9.1-9.3 (directed graphs)
__ AU sec. 6.1-6.5 (lists)
__ AU 5.2-5.3 (linked structures)
__ AU ch. 1 (data abstraction)
__ AU sec. 6.6 (stacks)
__ AU sec. 6.8 (queues)
__ LA pages 1-105 (classes and objects)
__ AU sec. 5.10 (priority queues)
__ AU sec. 4.4-4.5 (storage allocation)
__ LA 125-148 (storage allocation)
__ LA pages 106-124 (pointer arithmetic)
__ AU sec. 2.6-2.8, 2.2-2.4, 3.10, 5.4 (recursion)
__ AU 6.9 (dynamic programming)
__ AU sec. 5.6 (binary trees)
__ AU sec. 11.2-11.6, 11.8 (grammars)
__ AU sec. 2.5, 2.9 (correctness of programs)
__ AU ch. 3 (analysis of runtime)
__ AU ch. 7 (sets and dictionaries)
__ AU pp 290-291 (binary search)
__ AU sec. 5.8 (binary search trees)
__ AU sec. 7.6 (hashing)
__ AU sec. 3.9, 5.11 (sorting)
__ LA pages 149-168, 187-198 (customization, inheritance)
__ AU sec. 9.6-9.9 (searching, path-finding)
__ AU ch. 10 (regular expressions, pattern matching)
__ Numeric Computing handout
__ AU ch. 13 (logic gates, Boolean algebra)
__ AT sec. 4.1 (logic gates, Boolean algebra)
__ AU 12.6 (Karnaugh maps)
__ Hypercubes handout
__ Sequential Logic handout
__ AU sec. 4.1-4.3, 4.6-4.8 (processing unit structure)
__ AT ch. 2, 3, sec. 4.2-4.6 (processing unit structure)
__ ISC handout
__ AT ch. 5 (instruction sets and addressing modes)
__ AT ch. 7 (assemblers)
__ AT ch. 6 (virtual memory, operating systems)
__ AU sec. 14.10 (limitations)

(Other handouts to be introduced as necessary)

Honor Code Standard

We observe the following Standard: "You may *discuss* the assignment with other students currently in the course. You may not share written work of any kind, inside or outside the course". Elaborations: In the case of programming assignments, we consider computer files, fragments of files, and printed output to be "written work". In developing code for a programming assignment, you can discuss *ideas* with others. Once you have an idea, it is up to you to develop it yourself. Therein lie the key learning experiences of the course; don't give them up. Discussion of ideas must not involve transcription of the actual working code of others. *Definitely forbidden* is a common (only elsewhere, hopefully) form of collaboration wherein two or more students split up an assignment, then transcribe each others' contributions, sometimes changing names of variables. Programming in CS60 emphasizes individual contributions, rather than group projects. If the help you get from another is significant, you should acknowledge it on your submission. If you have any doubts about whether a form of interaction constitutes a violation of this standard, please consult with the instructor *prior* to continuing.

Programming Assignment Topics

The programming assignments provide one of the main learning experiences in the course. Each one deals with one or more key points and is intended to help drive home the understanding of those points. Not all assignments are weighted the same. Weight depends on difficulty, and the final weighting is not determined until the end of the term. In this offering, the anticipated assignments and some of the points emphasized are:

1. Touring with Turing
   - Turing machines and their programming
   - The state concept
   - Use of programs as subroutines
   - Fundamental theorem of arithmetic and prime numbers
2. Life on a Donut
   - Systems of synchronous state machines
   - Programming in C++, using arrays
   - Procedures as parameters
   - Self-replication
3. RPN (Reverse Polish Notation) for Fun and Profit
   - Use of stacks
   - Building an RPN calculator
4. Data Abstraction Under the Hood
   - Implementing (rather than using) data abstractions
   - C++ class definitions
5. Bignums – How High Can You Go?
   - Number representation
   - Arbitrary-precision integers
   - Program speed measurement and analysis
6. Fun with S expressions
   - Scheming Up List Structures – An abstraction for open lists
   - Recursive programming
   - Polymorphism
   - A Boolean Expression Simplifier
7. Culinary Techniques for Fast Spell Checking
   - Set and Mapping abstractions
   - Hashing
8. From Food to Wine – A Word Game
   - Graphs
   - Path-finding
9. Playing the Numbers – Numeric Programming
   - Equation solving
   - The hazards of roundoff
10. Machine-level Programming on the ISC (Incredibly Simple Computer)