Esoteric Programming Languages and Algorithms: Blum-Blum-Shub and other topics

Student Taught Course, spring 2010

Student instructors: Eric Stansifer (senior, Mathematics) and Nathan Watson (sophomore, Computer Science)

Sponsoring Professor: Leonard Schulman

Course description:

The goal of this course is to introduce the student to ten programming languages and algorithms that are not covered by typical computer science curriculum. We aim to bring the student to a basic level of proficiency in each language, and to a general understanding of each algorithm discussed. Over the course of the term, a number of unusual programming techniques, theoretical ideas, and clever tricks will be shown to the student.

Students will be introduced to five languages and five algorithms during this course. The algorithms component of the course will cover a number of advanced techniques used by current researchers, as we discuss papers on efficient convex-hull and matrix multiplication algorithms. Pseudo-random number generation will be discussed in the course. Each of the languages in the languages component is in some way innovative or fundamentally different from any commonly used general purpose language today. For the FP and J languages, the student will be introduced to the techniques of functional programming, and several theoretical topics including type theory and calculus of types will be covered. FP is a historically significant language, invented by Turing Award winner John Backus in 1977 to demonstrate how functional programming permits programs to be algorithmically manipulated, a key innovation of the time that has profoundly influenced modern functional programming. Notions such as strict, eager, lazy, dynamically-typed, and statically-typed languages will be discussed in lecture. Through exposure to a diversity of fundamentally different languages, students will learn key elements of programming language design and be better able to critique and effectively use the more mainstream languages. None of the topics addressed in the course are covered in the Caltech CS curriculum.

Format:

The class will meet for a single three hour meeting, once a week. The first hour will be a lecture on that week's topic, and the remaining time will be devoted to completing the class assignments. Students will be encouraged to work together and ask the instructors for help.
It is expected that students taking this course are competent programmers and able to write small programs independently. Students who have performed well in CS 1 should have sufficient background to take this course; past or concurrent enrollment in CS 21 or 38 is helpful but not required. While not much theoretical background is required, familiarity with big-Oh notation is assumed. Students will be given a choice of assignments to complete, so as to better

accommodate those of different skill levels. The more difficult assignments may require advanced theoretical knowledge of programming techniques.

In odd weeks, assignments will consist of a number of small programming exercises in that week's language; students will be expected to demonstrate a basic level of proficiency in the language to pass. In even weeks, assignments will consist of larger programming tasks connected to that week's algorithm, which can be completed in a language of their choice; students will be expected to demonstrate familiarity with the essential concepts of that algorithm to pass. Students must achieve passing grades in four of the five languages and four of the five algorithms to pass the course as a whole. The languages will be taught in odd weeks by Eric; the algorithms will be taught in even weeks by Nathan.

Topics by week:

1. Befunge. A Befunge program is a two-dimensional grid of instructions; program execution walks along the grid in straight lines, which can be deflected by various commands. Befunge programs may be self-modifying. Befunge is a stack-based language.
2. Convex hull (Jarvis, Graham, and Chan). We first discuss the standard gift-wrapping algorithm for computing convex hull in two dimensions. Afterward we discuss an algorithm that achieves the theoretical optimal bound – the execution time is determined by the size of the output.
3. POV-Ray SDL. The POV-Ray Scene Description Language is a computer graphics language, used by the POV-Ray renderer. However, the macros in the SDL are powerful enough for the language to be Turing complete; we give an example where a renderer was written in the SDL.
4. Generating permutations (Lexicographic order and Steinhaus-Johnson-Trotter). After presenting the canonical lexicographic ordering of permutations, we discuss the Steinhaus-Johnson-Trotter ordering and an algorithm to generate it that surprisingly works.
5. FP. FP ("Function-level Programming") was invented in 1977 by Backus to demonstrate the potential for programs to be mechanically manipulated by computers, for example, by compilers to optimize a program without changing its behavior. However the language – never meant for practical use – entirely uses "point-free", aka "point-less", style and lacks input and output.
6. Pseudo-random number generators (Mersenne Twister and Blum-Blum-Shub). As Knuth famously said, writing a random number generator is anything but random. Modern random number generators use highly sophisticated techniques to ensure randomness; the Mersenne Twister is one of these.
7. J. J, which is based in part on FP and APL, is an obscure scientific computing language that has achieved cult status and is still used seriously today. The language is extremely terse yet powerful, and has excellent built in support for matrix manipulation and complex and rational numbers.
8. Multiplying matrices (Strassen, Coppersmith-Winograd, and Group theoretic approach). Fast matrix multiplication is essential to all forms of numerical computing. We discuss a series of successively faster algorithms, including one which (if a particular conjecture is true) achieves O(n2).

9. Postscript. Postscript files are most commonly computer-generated (for example, by TEX), to describe text layout on a page at a low level. However, Postscript is a very simple stack based language that can be directly written by humans, most famously for high quality fractal generation. We will discuss the programmatic aspects of the Postscript language.
10. Queues (Brodal queues) and linked lists (Xor-linked lists). Brodal queues achieve guaranteed time bounds that Fibonacci queues merely asymptotically approach; xor-linked lists cleverly save on memory.

Qualifications of instructors:

Eric and Nathan have each represented Caltech at the world finals of the International Collegiate Programming Competition, and each has coached the Caltech team for the same competition. Coaching requires running the Caltech intramural competitions, and planning and organization over the course of a term; it also includes a teaching component. Nathan has taken CS 38 (an algorithms class) and Eric has taken CS 151. Both have an avid interest in unusual programming topics and have explored such interests outside of class time.