Introducing Computer Science After Programming

Cary G. Gray
Wheaton College
Wheaton, Illinois
Cary.G.Gray@wheaton.edu

Michael D. Frazier
Abilene Christian University
Abilene, Texas
Mike.Frazier@cs.acu.edu

ABSTRACT

We describe replacement of a more traditional (CS1/CS2) introductory sequence with one that starts after a one-semester programming-only course. The change simplifies placement for entering students (whether from other colleges or directly from high school), and the new structure more effectively communicates the nature of the science to beginning students. The new sequence accommodates the entering students who lack a high level of mathematical maturity, but without compromising the mathematical nature of the major.

1 INTRODUCTION

The introductory computer science sequence at Abilene Christian University (ACU) had evolved from the old programming-first model, adding survey material to CS1 in an attempt to give students a real taste of computer *science* early. Others have noted similar concerns [HKK99, TBB98] and proposed a variety of structures to meet them, particularly following the suggestions in [ACM91]. One of the key questions has been how to relate the introduction of computer science as a discipline to initial instruction in programming.

ACU has fairly open admissions standards, and many would-be computer science majors arrive possessing a low level of mathematical maturity, though many have prior programming experience. Since the fall of 1999 we have been using a novel course structure that places the introduction to programming *before* the first course in the major. While this structure was developed to deal with a specific student population, we have found that the resulting structure offers significant advantages that make it more broadly useful.

We describe in Section 2 the background and analysis that led to our proposal. The new sequence is described in detail in Section 3, and our experience with it is evaluated. Section 5 compares this structure with others and highlights why a similar arrangement may be appropriate even for very different student populations.

CS1	CS2	Algorithms
programming 1		
recursion		
	programming 2	
survey		
	elem. algorithms	
		int. algorithms
		adv. algorithms

Figure 1: Old course structure

2 BACKGROUND AND ANALYSIS

ACU's previous introductory sequence

ACU offers a CS major that is intended as a *science* major—we want our students to learn to think like scientists. Discrete Math is required early in the program so that we can cover material more deeply (and efficiently) in appropriate major courses.

ACU's intro sequence had evolved from the classic (imperative) "programming first" model, in which the two semesters of the first year concentrated on programming. We had previously found ourselves struggling with the "computer science is programming" misperception, and so we had added to the first semester a significant effort to survey computer science, as in, for example, [HS97]. The second semester included the beginning material on analysis of algorithms, supported by a Discrete Math corequisite.

Figure 1 shows the allocation of material among these two courses and the junior-level Algorithms and Advanced Data Structures which was required of all CS majors. The modules in the figure, with rough description in terms of CC2001 [ACM01] knowledge units,[1] are:

programming 1 introductory programming and problem solving, using procedures/functions and static data structures (PF1, PF2, and about half of PF3)

recursion first taste of recursive functions (most of PF4)

programming 2 continued development of programming skill, including dynamic (pointer-based) data structures, with strong emphasis on abstraction and modularity (remainder of PF3 and PF4, PL5, much of PL6)

survey a survey of major areas in computing and computer science; not exhaustive, but with enough breadth to be representative (portions of AL2, OS1, NC2, PL1, PL2, PL3, SP1, SP2, SP4–7, AR3; most or all of AR2, AL5, IS1)

[1] This work was done prior to publication of CC2001, so that our thinking more naturally maps onto CC'91 [ACM91], but we hope this description is more useful.

elementary algorithms first work with analysis of algorithms, typically via sorting and searching (most of AL1, about half of AL2 and AL3)

intermediate algorithms more advanced algorithms and their analysis; the balance of the algorithmic material essential for every computer science major (balance of AL1–3, AL6, part of IS2)

advanced algorithms additional algorithmic material suitable for elective coursework (including AL8, AL10)

A significant number of students arrive at ACU with unrealistic expectations about our major. We therefore put special emphasis in the first semester on setting expectations for the major through a survey of computing broadly and computer science in particular. We want students who are not interested in our offerings to move quickly to a more appropriate major so that they can graduate in a timely manner. Requiring discrete mathematics early in the course sequence also helps to set realistic expectations.

ACU also offers a service course, Introduction to Scientific Computing, as a one-semester introduction to programming and other issues for majors in the hard sciences. This course has often been taught by faculty from another science department, for whom Fortran is the preferred language. The programming material covered corresponds to "programming 1" in our list of modules; the course also addresses other issues that arise in numerical calculations.

Student population

Most students enter ACU as freshmen, directly from high school; the university as a whole receives very few students who have completed a two-year program at a junior college. Many of our entering students have had some instruction in computer programming: many have had a course in high school; others have completed a single course at a junior college. While many of the high school courses qualify for credit through the Advanced Placement program, almost all of our students have taken the Computer Science A exam rather than the more advanced Computer Science AB. As a consequence, the students with previous computing have typically been exposed only to programming, and typically to no more programming than we cover in the first semester.

In addition, admission to ACU is fairly open, such that many students arrive with limited mathematics background. A little over half require at least one semester of mathematics before they can (or should) take Discrete Math,[2] and a significant number require additional remedial courses. The Discrete Math corequisite to CS2 is a significant barrier for these students.

A number of students express interest in taking computing coursework—perhaps as much as completing a minor—but are not interested in the computer *science* major. We would like to serve these additional students, providing we can do so without compromising offerings for our majors.

[2] At ACU, the prerequisite to Discrete Math is Precalculus, though we recommend that students without calculus credit complete a semester of calculus before taking Discrete Math.

Motivation for change

Experience in teaching these courses, advising entering students, and dealing with transfer courses revealed several problems:

Placement from high school
Placement of entering students is a problem. Students come from high school with widely varying programming backgrounds, but very few students have significant exposure to the other material we cover in our introductory courses. The single greatest challenge of the first semester is grasping the process of creating a program, but a significant number of students in the class already have mastered that challenge. This leads to two problems: boredom for the more advanced students and intimidation for the less advanced.

Transfer articulation
Evaluation of transfer coursework was critical to determine whether the transferred course included enough of the survey material from CS1 to be considered equivalent; in most actual cases it did not. The catalog description alone was often insufficient to make the determination.

Internal articulation
Articulation is awkward for students who change majors to CS after taking Intro to Scientific Computing. Like students with programming experience from high school, they have mastered much of the programming content of our first semester, but often do not have all of the non-programming background required by our second semester.

Course focus
Each of the first two courses has two goals, but students have difficulty keeping both in mind. Many students in the first semester are so overwhelmed by learning to program that they miss the significance of the survey material. (This is further evidenced by students' persistence in referring to CS1 by the name of the programming language used.) Similarly, the programming material in the second semester tends to crowd out in the students' minds the deeper material on analysis of algorithms.

Discrete Math as a barrier
Almost all advanced CS courses build on material from our second semester, but the Discrete Math corequisite closes that course—and all subsequent courses—to most ACU students.

3 THE NEW PROGRAM

We have addressed these problems by reallocating material from our first two semesters—plus the junior-level Algorithms course—among three new courses:

Introduction to Computers and Programming
A beginner's course in programming and problem solving, supplemented by additional material on topics such as history and social issues to make it valuable to a student who does not take additional CS.

Fundamentals of Computer Science
survey of computing disciplines; object-oriented design, abstract data types, and fundamental data structures
prerequisite: Intro to Computers and Programming, or equivalent

Intro to Programming	Fundamentals of CS	Elem. Algorithms
programming 1		
	recursion	
	programming 2	
	survey	
		elem. algorithms
		int. algorithms
enrichment material		

Figure 2: New course structure

Elementary Algorithms and Data Structures
analysis of algorithms and data structures
prerequisite: Fundamentals, Discrete Math

The allocation of material among these new courses is shown in Figure 2.

The core of Intro to Programming is rather unambitious, leaving out the introduction to recursion (and even a taste of pointers) that was included in our old CS1. The essential requirements are low enough that we seldom have to worry about whether transfer work is adequate to substitute for it.[3] The instructor has considerable freedom in choosing material to flesh out this course; the primary concern in that selection is serving those who will not go on to more advanced work.

Each of the problems noted in the previous section is addressed by this restructuring:

Placement from high school
Students with adequate programming background can be placed directly into Fundamentals, which does not require the survey material that few high schools (and the junior colleges we see transfers from) teach. Students with little or no programming background spend a semester in Intro to Programming focused on that one issue.

Transfer articulation
Granting transfer credit for the new Intro to Programming is much less risky than for the old CS1. Transfer students with a stronger first semester course face a mix of new and familiar material in Fundamentals of CS.

Internal articulation
Intro to Scientific Computing, without change, now substitutes well for Intro to Programming.

[3] On the other hand, few of our students transfer out after just one semester of CS.

Course focus

The first and third semesters each have a single focus, clearly reflected in their titles. The two components of Fundamentals of CS are more nearly balanced than in the old CS1.

Discrete Math as a barrier

Prerequisites for CS courses that had listed CS2 (and therefore implicitly Discrete Math) have been reevaluated to identify those that require only Fundamentals; courses that required the old advanced Algorithms have changed to require only the new Elementary Algorithms.

The main drawback of this structure is that it expands our introductory material from two semesters to three for students who do not have previous programming experience. For CS majors, the elimination of the advanced Algorithms course as both a requirement and a prerequisite offsets the increase, so that the total number of semesters in sequence remains the same. Also, because Fundamentals of CS is the key prerequisite course, students can begin some more advanced work in the same semester that they are taking Elementary Algorithms.

The new arrangement does still present an articulation problem for students who enter ACU with more than a semester of college-level computing. Such students might be either transfers from other colleges (especially those with a traditional CS1/CS2) or freshmen from a very strong AP Computer Science AB course (with good coverage of analysis of algorithms). We see few such students, however, compared to the number who were poorly placed in our old course sequence.

With the new curriculum, both the CS major and minor begin with Fundamentals of Computer Science. The Intro to Programming course does not appear as a requirement in either program.[4] We have come to view Intro to Programming as analogous to Precalculus in a mathematics program: a large share of students are getting this material in high school, but it is far from universal. We therefore provide a leveling course before the major for those who need it.

The placement of Discrete Math in the prerequisite structure is significant. In particular, Discrete Math no longer lies on the path to *all* advanced courses in the CS program. This removes an accidental barrier that has kept math-averse students from taking additional courses. We hope that this will allow us to offer a minor in Applied Computing to serve those students' interest in computing, without diluting the experience that we (and others [TKB01]) believe is vital to a major in computer science.

4 EXPERIENCE

The new curriculum has been in place for two years. Several of the expected advantages show up in advising and administrative contexts:

- The catalog text—and especially the course titles—is more supportive of our "computer science is not the same as programming" message. This message is

[4] Intro to Programming does count toward the total required hours in CS, so that the total hours of CS required has not increased. Students who begin in Fundamentals take an additional advanced course instead. Previously, a petition to substitute more advanced work was required for those students whom we placed out of CS1; the new structure has eliminated the need for those petitions.

further reinforced by the fact that Intro to Programming is not listed as a major requirement.

- Placement into the correct introductory course has been greatly simplified. Most students can be correctly placed after a brief interview. The first programming activity in Fundamentals of CS helps to settle the few students whose initial placement is incorrect (with the scheduling of Intro to Programming and Fundamentals of CS arranged to facilitate the move).

- Evaluation of transfer coursework has been much easier; the more difficult decision of when to consider transfer work as equivalent to Fundamentals is rare.

Benefits have also been apparent in the classroom:

- The twin problems of intimidation and boredom are noticeably decreased, because students in each class are closer to the same level. The reduction in need to explicitly compensate for intimidation of students without prior programming experience is especially noticeable.

- All three of the new courses seem much more coherent than our old CS1/CS2.

In addition, we are able to make much more use of Discrete Math as a *pre*requisite to Elementary Algorithms than we were with it as a *co*requisite to CS2. That delay also gives students a little more time to mature mathematically.

Fundamentals of CS remains a rather ambitious course, with its dual goals reflected in the use of a pair of textbooks, one for survey material (such as [Bro97]) and the other for the programming material. Two aspects of our approach to this course help it to cohere, even in the minds of students: the programming exercises are chosen to work with the key ideas in the survey, and the entire course is structured around the theme of *representation*.

The theme of representation encompasses more than the encoding of data. It includes representation of programs, including the idea of levels of abstraction and virtual machines. Analysis of the performance trade-offs of different data structures demonstrates the significance of choice of representation. Different classes of automata represent computational models, seeking a minimal representation that achieves universality—and providing opportunities to show equivalence of representations that appear quite different. Representation is a unifying theme that goes to the heart of computer science.

Using programming in treating the survey material allows a richer treatment. When representing numbers, students write programs to interpret binary strings as twos-complement numerals. Instead of using a provided simulator for automata, students write and then use their own simulator. Students set up their own experimental comparisons of sorting algorithms by producing programs and instrumenting them, then collecting and analyzing the data.

The assignment that students universally regard as the most challenging is the implementation of a simulator for a simple machine language [Bro97, Appendix C]. Writing this program helps the students appreciate how much they take for granted when working problems by hand, as opposed to fully specifying their thinking to a computer. This assignment drives home the importance of abstraction as it gives students a concrete example of its power.

This greater depth does mean that not as many topics can be surveyed; some areas must be left out. In spite of this, we think our introduction gives our students a very good sense of what it is to *do* computer science, to think like a computer scientist. We are quite happy to trade a bit of breadth for the depth of experience gained.

One goal for restructuring the introductory courses was to open additional courses to non-majors, with the hope of offering a minor in Applied Computing. Discrete Math is no longer on the prerequisite path to many CS courses, particularly those in software development and in computing systems. The way is now clear to a minor with a reduced mathematics requirement, but we have not yet made progress toward that end.

5 COMPARISON WITH OTHER STRUCTURES

CC2001 [ACM01] describes several structures for introductory sequences. We had already found that the "programming first" structures (whether imperative or object-based) did not serve our program's goals. There are structures that manage to integrate deeper concepts with introductory programming, as exemplified especially by [HKK99] or even [AS96], but we believe that many of our students would be overwhelmed by the abstract thinking required if they encountered this material as freshmen. Given our student population, we need to bring them to that level by a more gentle path, and our more gradual approach gives us a chance to convince them of the value of abstraction.

The label "breadth first" is often applied to structures that aim to give students a good sense of the discipline early, emphasizing that computer science is not the same as programming (or even software development). While we heartily agree, we must point out that the label has been applied to rather different structures.

One interpretation [TG91] combines the introductory programming with survey material over two to four semesters. That is actually rather similar to our old program. For a diverse student population, it has the drawbacks we encountered with our old program.

Another "breadth-first" approach is to begin with a survey course, sometimes labelled CS0, taken before (or independent of) the first course in programming. We should note that different kinds of courses have been called CS0. Some survey courses are aimed principally at majors and so may also introduce professional issues; [Coo97], for example, emphasizes problem-solving as a prerequisite to programming. Others, such as [GBL94], are aimed at a broader audience, doubling as a service course—and a recruiting ground for potential majors. The common element is that these surveys do not require a programming background.

This approach would not work well for ACU's student population. The mathematical level of most entering ACU students is low;[5] teaching a CS0 with sufficiently low mathematical demands to make it accessible would, we believe, require watering down or eliminating some of the topics that are crucial in introducing

[5] And the level of aversion to mathematical material is high.

the discipline to majors. A too-shallow introduction does not provide a strong foundation; it could, in fact, be worse than no survey at all, if it does not prepare students for the rigor to come and leaves them with an inaccurate impression of the discipline. One of the authors now routinely teaches a no-programming-required survey course in another college; that course works at all only because the students there have a high enough level of mathematical sophistication.

CC2001 briefly mentions what it describes as "breadth second":

> Another approach to providing breadth in the curriculum is to offer a survey of the field after the completion of the introductory programming sequence. This "breadth-second" approach means that students begin with a programming-based introduction to make sure they have the necessary implementation skills but then have an early chance to appreciate the range of topics that are all part of computer science. ...

This comes closer to our model than anything else we have seen. Significantly, though, we place the survey material after just one semester of programming rather than a sequence of two or more courses. CC2001 continues:

> ...While we feel that such an approach is worth pursuing as an experiment, we have not yet found models that meet our criterion for acceptance.

Our program can be viewed as such an experiment. The initial results have been quite good, though with a small number of students.

CC1991 placed "Introduction to a Programming Language" as an optional topic, outside the core, assuming that "increasing numbers of students do gain such experience in secondary school" [ACM91, p. 18]. CC2001 retreats from that position, placing "Programming Fundamentals" within the core. We think that CC1991 had it basically right, but that the high-school experience of our students is too uneven to provide a solid foundation. By repackaging the courses so that Intro to Programming does correspond to what our students get in high school, we have been able to push that first taste of programming *before* the CS major. Because all of our students take Fundamentals of CS, that course ensures that all have the foundation they need for more advanced work.

We should note a parallel between our new structure and what is common in departments of mathematics. At one time, few high schools offered calculus, and many did not even offer a precalculus course. College mathematics programs therefore began with a pre-calculus course or, more recently, the first semester of calculus. As the level of high-school mathematics has risen for many students, mathematics departments have raised the level at which the major begins. We have placed the first programming course *before* the major instead of as the beginning of the major.[6]

Similarly, few mathematicians would consider the sort of algebraic manipulation that dominates pre-college instruction as really "doing mathematics", but learning

[6] Our move is counter to the shift in CC2001. The difference in direction may be accounted for by the low expectation we have for high-school programming instruction. CC1991 may have expected something analogous to a semester of Calculus, while we have aimed only for the Precalculus.

those skills is an essential prerequisite to eventually doing interesting mathematics. We take a similar view of elementary programming skill: it is essential, but it is not very characteristic of doing actual computer science.

The prerequisite of programming might seem at odds with recent recommendations regarding increasing the accessibility of computer science to underrepresented groups, especially women. CMU, for one, has aggressively promoted the fact that its CS major requires no programming background in order to increase participation by women [Blu01]. We have been very careful, however, to not extend the requirements for students who require Intro to Programming: it fills an elective slot within the major, and the course offerings are arranged to comfortably accommodate entering students who need that introduction. One of the motivations for our structure was to reduce intimidation by having students in each class at more nearly the same level, and the new structure has achieved that.

6 CONCLUSIONS

We have described a structure for the introductory course sequence that places the introduction to computer science after a one-semester introduction to programming, with the content of that semester matching programming courses of students entering from high school. Two years of experience with this structure has shown that it fits our student population quite well. Furthermore, using programming exercises while surveying the discipline enables a much richer—and consequently more authentic—treatment of the material. The arrangement of our sequence also provides our students with a gradual introduction to abstract thinking that appears well-suited to their needs.

REFERENCES

[ACM91] ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 1991*. Association for Computing Machinery, February 1991.

[ACM01] ACM/IEEE-CS Joint Curriculum Task Force. *Computing Curricula 2001: Computer Science*. Association for Computing Machinery, December 2001.

[AS96] Harold Abelson and Gerald Jay Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, second edition, 1996.

[Blu01] Lenore Blum. Transforming the culture of computing at Carnegie Mellon. *Computing Research News*, 13(5):2, 6, 9, November 2001.

[Bro97] J. Glenn Brookshear. *Computer Science: An Overview*. Addison-Wesley, fifth edition, 1997.

[Coo97] Curtis R. Cook. CS0: Computer science orientation course. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, pages 87–91, February 1997.

[GBL94] Michael Goldweber, John Barr, and Chuck Leska. A new perspective on teaching computer literacy. In *Proceedings of the Twenty-fifth SIGCSE Technical Symposium on Computer Science Education*, pages 131–135, March 1994.

[HKK99] Max Hailperin, Barbara Kaiser, and Karl Knight. *Concrete Abstractions: An Introduction to Computer Science Using Scheme*. Brooks/Cole Publishing Company, 1999.

[HS97] Geoffrey Holmes and Tony C. Smith. Adding some spice to CS1 curricula. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, pages 204–208, February 1997.

[TBB98] Allen B. Tucker, Keith Barker, Andrew P. Bernat, Robert Cupper, Charles F. Kelemen, and Ruth Ungar. Developing the breadth-first introductory curriculum: Results of a three-year experiment. *Computer Science Education*, 8(1):27–55, March 1998.

[TG91] A. Tucker and D.A. Garnick. A breadth-first introductory curriculum in computer science. *Computer Science Education*, 3:271–295, 1991.

[TKB01] Allen B. Tucker, Charles F. Kelemen, and Kim B. Bruce. Our curriculum has become math-phobic! In *Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education*, pages 243–247, February 2001.