CS2 ABSTRACTS

Keynote
Henry Walker, Grinnell College

Each course should have purpose, value, integrity, coherence, and intellectual excitement. At a very beginning level, CS1 can introduce a new world of problem solving and can help hook students, so they will want to explore the discipline of computer science further. At the upper levels, a course can focus on the challenges and opportunities of a specific topic (e.g., operating systems, algorithms, programming languages, HCI, etc.). In some settings, however, the goals, content, and pedagogy for CS2 may be fuzzy. For example, CS2 may be CS1++, and CS2 may serve more as a bridge than a legitimate, stimulating course in its own right. This talk will consider several alternative goals for CS2, together with ideas for content and supporting pedagogy.

Why Computer Games should be BANNED in the Undergraduate CS Curriculum
Henry M. Walker, Grinnell College

Computer games are inappropriate within the [introductory] CS curriculum for at least 3 reasons:

1. Any emphasis on computer games likely reinforces incorrect and counter-productive cultural stereotypes regarding the nature of computer science.

2. Games often discourage underrepresented groups, who actively seek out disciplines which are seen to help people's lives, yet these are the groups that are most actively sought to achieve diversity.

3. Games represent an extremely small part of the IT industry (well under 2% of gross revenues in the US), so an extensive focus on games can give students a distorted nature of the IT industry.

Rather than reinforce popular stereotypes and destructive myths, [introductory] CS courses should present concepts within the many interesting and socially-responsible contexts available. Catering to a niche market does not serve our students well.

Arguments in Favor of Games
Robert Keller, Matthew Kretchmar, and Jeff Parker

A Lab-based Introduction to Computer Science that Emphasizes Collaboration
Henry M. Walker, Grinnell College

In introductory computer science at Grinnell College, students complete about 47 laboratory exercises, and I lecture about 4 hours per month (mostly in 5-10 minute segments). Students prepare for most class meetings by reading new materials (available on the Web). During class, students work in pairs at a workstation on exercises that involve solving problems, writing new code or revising existing code, experimenting, and writing explanations. The person at the keyboard changes at least daily, and I assign new partners weekly. As teacher, mentor, and coach, my role is to visit student pairs frequently to clarify ideas, identify difficulties, and offer directions for solution. In addition to working collaboratively on labs and lab write-ups, students work individually on selected programming problems (assigned as homework) and on tests. Altogether, the approach pushes actively learning to an extreme, and our experience suggests that this pedagogy allows us to cover about 20% more material than our traditional approach (with separate lectures and labs), and our students perform better on tests. The approach also seems to help student recruitment and retention. On the down side, an instructor beginning this approach must develop (write, borrow, adapt) all 47 labs.

Twofold Approach to CS2
Benjamin Kuperman, Oberlin

My approach to CS2 is twofold. First, I want the students to understand and appreciate the differences between various data structures (and associated algorithms) based on run-time complexity. I want them to learn these concepts as general concepts, not tied to one language in particular. Therefore, most lectures are done without any code whatsoever. I also am constantly asking "What is the runtime complexity?" but more importantly, I insist that they be able to explain the ***why*** behind their answer.

The second part of my approach is based on my belief that to truly learn the material, you have to experience it. My classes are constructed around 10+ programming assignments, each focusing on some data structure. With the speed of computers increasing so rapidly, I try to build the assignment around large, real-world data sets. The assignments I'll demo are an Email Directory using all students at the college (LinkedLists), a version of the Kevin Bacon game using all of IMDB (Graphs), and a 3-part search engine assignment using all the web pages of the dept., college, and beyond (BST, Heaps, Hashtables, GUIs).

A Multi-Lingual Approach to CS2
Robert M. Keller, Harvey Mudd College

Students embarking on CS 2 already know what programming is about. What they might not appreciate is the range of paradigms that arise in programming and in computer science in general. Our view is that while it is possible to demonstrate these paradigms in a single language, it is more efficient and directed to use languages that are designed for the purpose. To this end, we have been using 3 to 4 languages in CS 2 for the last 15 years. I will describe our current approach and elaborate on the rationale. The following table illustrates the kinds of languages we use at Harvey Mudd College.

Paradigm	Language(s)
General background	Python
Functional programming	Python, Scheme, Java
Programs as data, higher-order functions	Scheme
Object-oriented programming	Java
Logic programming, databases	Prolog
Backtracking, constraint-directed programming	Prolog
Grammar-directed interpretation and translation	Scheme, Prolog
Automata models (finite-state and Turing machines)	JFLAP, Prolog

As much as possible, we would like for the students to gain the perspective that all languages are equivalent at some level, but that the choice of language can make a significant difference in the efficiency of development and clarity of presentation. The emphasis of our course is hands-on problem solving using weekly exercises involving these concepts and their connections.

Tool Demo
Brian Howard, DePauw
I would like to demo the tool my students and I developed for exploring recursion with a language similar to Haskell. We have been using it in our CS2 class for the past five years, and it has worked pretty well for us.

Teaching Recursion
Jeff Parker, Merrimack

A look at some of advice we have seen and some interesting examples and assignments. Students find recursion hard to understand, and faculty find it essential to teach. How can we convince students that recursion is helpful? In this talk, we look at a number of suggestions from the literature, and look at sample projects that can be given in CS2, as well as in other courses. We will look at the following approaches used in class and on homeworks:

JTS - why didn't you Just Teach Scheme in the first place?
Graphical - great motivating examples

Textual - String manipulation
Textual-Graphics - letter pyramids, parsing parens
Games - Towers of Hanoi, rings and things
Interactive Games - Games they can play while sitting down
Exercises - collection of programming exercises