PANEL PROPOSAL:

HOW SHOULD THE SECOND COMPUTER SCIENCE COURSE (CS2) BE TAUGHT?

James Aman,
Department of Mathematics and Computer Science
Wilmington College

Richard Close,
Department of Science and Associate Dean
United States Coast Guard Academy

Danny Kopec,
Department of Computing, Mathematics, and Science
Richmond, The American International University in London

James Aman,
Associate Professor of Computer Science,
Wilmington College

The CS2 course at Wilmington College is an intensive study of a high-level programming language. For several years we have used Pascal in this course and will probably continue to do so in the foreseeable future. Besides the obvious need for instruction in the specifics of the language, this course carries strong emphases on
fundamental algorithms and basic software engineering design techniques. Study of algorithm development is continued from a strong basis laid in the CS1 course.

Of particular concern in the design of this course is the selection of a good textbook. Certainly there are more Pascal books available than there probably need be. Most are carefully designed and well written. But we am looking for a particular characteristic difficult (in our estimation) to find: cohesion among problem sets. Experience over the past decade tells us that students do better work when the

programming assignments represent progressive refinements of a few problems. Our objective is to find a book that adopts this case study approach.

The problem with this quest has been striking a balance between good (or acceptable) problem sets and implementation-appropriateness. In our case this latter term means having a book that deals with Turbo Pascal for Windows, rather than just Pascal. The book also needs to place emphasis on the particular set of issues, techniques, and skills we consider most appropriate. The search is never perfect, but it continues.

Richard Close, Professor,
Department of Science,
United States Coast Guard Academy

CS2 is always changing; a moving target. CS2 was first introduced at the U. S. Coast Guard Academy under the title of Technical Programming almost 20 years ago. At that time, it had been noticed that CS majors really did not become proficient programmers in CS1, so a more thorough grounding in programming concepts was offered along with a good dose of theory - data structures and some algorithm analysis. Informally, the course often was known as "Baby Data Structures". This indicated that much of the material would be repeated - albeit at a more sophisticated level - in a subsequent course.

The original language used in our CS2 course was Pascal. Later C replaced Pascal because the prevalent feeling was that real programmers didn't use Pascal and in any case, a CS major should know more than one programming language. C turned out to be a little harder for students than Pascal, so some of the theoretical material was dropped. When object-oriented concepts surfaced, it appeared that it would be an easy transition to C++. Again, it was harder than it originally appeared and even less theory was included. Lately Java or Visual Basic seems to be gaining favor. This choice of programming language has made the course more attractive to non-majors, particularly engineers and CIS majors. However, reports from instructors in such courses indicate that the theoretical CS content is almost gone. It has become a programming course. This possibly relates to the popularity of the supposedly practical rather than theoretical aspects of the discipline. Many students (and faculty members) recognize that knowing Java and/or Visual Basic may make them more employable.

So, what should be included in CS2? Should we play to the audience and teach what a fairly large number of students seem to want? Or, should we continue with the latest ACM/IEEE recommendations and risk becoming extinct?

Danny Kopec, Senior Lecturer (Associate Professor),
Computer Science, Systems Engineering, and Management,
Richmond, The American International University in London

After the introductory programming course, (CSI), where students learn the foundations of programming through sound, structured, top-down techniques, the issues of how to best proceed with computer science programming instruction arise.

Bigger, modular problems illustrating various themes fundamental to the design and use of functional programming are posed. Typically critical are issues centered around the topics: parameter passing, scope and recursion. Then our course shifts to the study of arrays, fundamental data structures (stacks and queues), followed by basic sorting and searching algorithms. The course is completed with the study of pointers and file structures.

In CS1 and CS2 many instructors will assign diverse programming work of a theoretical nature. Such assignments may do well to illustrate the difficulties for a particular language to handle certain I/O constructs or to perform (or implement) operations on certain data structures. For example in CS2 students may be asked to handle character manipulation or to perform operations on arrays, stacks, or queues.

I believe that it is time for instructors to concentrate more on presenting programming problems that are of a practical value to students. Problems which need solutions to be programmed (as opposed to using one of thousands of applications available today – for example spreadsheets or databases) and which may benefit from have their solutions programmed. Students will thereby better understand the purpose and value of being able to program. Students' efforts in solving such problems will accomplish many important goals: 1) they will provide personal satisfaction in using what is learned in the classroom for personal convenience 2) students will develop a hands-on insight into the issues of problem solving from a choice of a variety of possible approaches and their tradeoffs. 3) students will be confronted with and will have to learn to deal with language-specific issues relevant to their problems. Specific examples will be presented.