

First Year Instruction in Computer Science: A Position Statement

Robert McCloskey
Department of Computing Sciences
University of Scranton
Scranton, PA 18510
mccloske@cs.uofs.edu

This short statement touches upon two aspects of first-year CS instruction. My opinions on these issues are well-represented (and influenced) by the works listed in the references.

1 Reasoning about Programs

If my experience is typical, one of the most glaring deficiencies among computer science students is their inability, when presented with a subprogram or short code segment (even one of their own creation), to explain *why* it works correctly. This, I believe, goes a long way toward explaining why so many students have difficulty in developing correct programs. In part, this deficiency is due to the fact that most students are never forced to use—or even taught—the mental tools necessary for devising such explanations. As a result, these students are doomed to rely solely upon intuition, unable to utilize well-known (and fairly accessible) formal methods that can aid a programmer not only in demonstrating a program’s correctness but also in developing the code.

CS educators should strive to ensure that all senior-level students have the ability to express, with precision, the intended effect of a subprogram (or code segment) via pre- and post-conditions and to annotate their code with meaningful assertions (e.g., loop invariants) that serve as a sketch of a proof that the subprogram (or code segment) has its intended effect (i.e., is correct). Talented senior-level students should, in addition, be able to devise rigorous proofs of correctness and even be able to develop a code segment simultaneously with its proof of correctness.

In order to achieve these goals, it is imperative that we begin laying a foundation during the first year of instruction. A rigorous course in discrete mathematics, including heavy doses of logic (both propositional and predicate) and mathematical induction, should be mandatory. (Preferably, such a course would follow the “calculational” approach taken in [3].) Instructors in the CS1 and CS2 courses should utilize these concepts whenever possible, illustrating their usefulness in both developing and verifying programs. It is even possible, as reported in [1, 8], to require freshmen students to develop programs using a formal approach.

2 Programming Language

In recent years, a number of CS educators have pointed to the widespread industrial use of languages such as C, C++, and Java in order to justify their use in freshmen-level CS courses. This is a mistake. While one of the goals of a CS program may be to prepare its students for

industrial software development, it does not follow that the programming languages currently popular in industry are appropriate for use in freshmen-level courses. Evidently, this was understood during the 1980's, when Pascal, a language designed specifically for teaching the principles of structured programming, achieved great popularity in academia. Unfortunately, it seems to have been forgotten during the 1990's.

Now that we have advanced from structured programming to object-oriented programming (OOP), we are in dire need of a programming language for teaching the principles of OOP to beginners.¹ In [6], Kölling identifies the characteristics of such a language and argues that Smalltalk, C++, Eiffel, and Java all depart too far from the ideal to be considered well-suited for use in CS1. He goes on (in [7]) to describe a programming language *Blue* (of which he is co-designer) that, he claims, *is* well-suited for use in CS1. I have no experience using Blue and thus cannot say whether it achieves its goals; however, it appears to be a laudable attempt at filling a void that has been left empty for far too long.

References

- [1] Denman, R., Derivation of Recursive Algorithms for CS2, *SIGCSE Bulletin* 28:1 (March 1996), pp. 9-13.
- [2] Gries, D., The Need for Education in Useful Formal Logic, *IEEE Computer* 29:4 (April 1996), pp. 29-30.
- [3] Gries, D. and Schneider, F., *A Logical Approach to Discrete Math* Springer-Verlag, 1993.
- [4] Gries, D., Teaching Calculation and Discrimination: A More Effective Curriculum, *CACM* 34:3 (March 1991), pp. 44-55.
- [5] Kelemen, C. (ed.), Astrachan, O., Baldwin, D., Bruce, K., Henderson, P., Skrien, D., Tucker, A., and Van Loan C., Computer Science Report to the CUPM Curriculum Foundations Workshop in Physics and Computer Science, October 28-31, 1999.
- [6] Kölling, M., The Problem of Teaching Object-Oriented Programming, Part 1: Languages, *Journal of Object-Oriented Programming*, January 1999, pp. 8-15.
- [7] Kölling, M., The Blue Language, *Journal of Object-Oriented Programming*, March/April 1999, pp. 10-17.
- [8] McLoughlin, H. and Hely, K., Teaching Formal Programming to First Year Computer Science Students, *SIGCSE Bulletin* 28:1 (March 1996), pp. 155-159.

¹This assumes that it is appropriate to teach OOP to beginning programmers. I am not convinced of this but am willing to concede it here.