

# Ada95: Still a Language of Choice for CS1 (and CS2)

Richard G. Hull  
Department of Computing Sciences  
Lenoir-Rhyne College  
Hickory, NC 28603  
dickhull@lrc.edu

## Introduction

As our profession has matured, a common consensus concerning what our students should learn in the first course sequence has evolved. It is widely agreed that our students need to understand the basic concepts of computer science, such things as basic problem-solving skills, the foundations of software engineering, algorithm development, basic theory, abstraction, and an introduction to rigorous programming, and modeling. However, how these topics are taught continues to divide the profession. The debate often revolves around what language, if any, to use in the beginning courses for Computer Science majors, and when to introduce object-orientation.

The level of professional maturity of incoming students compounds the problem. We, as educators, choose a language in which students can learn how to use the concepts that we teach. However, students seem to believe that the course is a programming course in the language and often get so involved in the syntax that they miss the basic concepts. The proper choice of a language can help reduce the discrepancy between what we, the educators, think we teach and what students think they are learning.

## The Nature of CS-1 and CS-2

In discussing the introductory Computer Science sequence, we should look at the answers to two important questions:

- 1) Who is the target audience? (How can we describe the students we expect?)
- 2) What is the desired outcome?

Taking the second of these questions first—it is generally agreed that the first sequence is not intended to simply teach programming, but rather to present the underlying concepts of computing. Rather than *train* our students in some chosen programming language, we desire to instill an understanding of the problem-solving and programming process, and to *educate* the student in the foundations of the field. We stress algorithmic thinking, design, abstraction, the role of computing in society and its limitations, and the conceptual and intellectual foundations of the discipline. The language should in some way be secondary—it will be used to illustrate concepts and to provide students with a platform on which to test their understanding. In choosing a language, we should select one that supports the concepts we intend to present—abstraction, data modeling, a disciplined approach to programming, and clarity.

Regarding the first of these questions—since this is an introductory sequence it seems appropriate to assume that incoming students have little or no programming experience (cf. M. Feldman, *Inspiring Our Undergraduate Students' Aspirations*, SIGCSE Bulletin, vol. 31, no. 2, June 1999, pp.4-7). The language we choose should be writeable and readable, and should guide the student in her travels through the learning process. These days many students come to us with a desire to learn the latest “hot” language, in hopes of getting part-time or summer jobs right away. But if we accept that these students are generally novices, do we really expect them to go out and program right away, and to do it well? I hope not. So rather than use the “language du jour” (as some have called it) for the wrong reasons, we should choose the language that best fits our course goals.

## Why Ada?

The languages we currently find in common use in CS-1 & 2 seem to be Java, C++, and Ada95. All three of these languages provide support for data abstraction, object-orientation, and modular programming. But at Lenoir-Rhyne we feel that Ada95 provides the best support for the student in many ways. Ada95 is a highly disciplined language, which requires a high amount of discipline on the part of the programmer. This can be a great benefit to the new programmer. The strong typing requires students to understand the nature of the program's objects and the way they will be used before generating the actual code. The many syntax checks performed by the compiler provide guidance to the student who is not sure of her direction.

We then present C++ and Java in the sophomore and/or junior years to students who are then better prepared to deal with the syntactic complexity and ambiguities of those languages. In private conversation at SIGCSE2000, John McCormick related the following:

A few years ago a recruiter came to the State University of New York (SUNY) at Plattsburgh looking for C++ programmers. But the Plattsburgh graduates had not studied C++. Their experience was with Ada. Even so, 15 Plattsburgh graduates were hired, as part of a total of 150 new hires by the company. One year later 14 promotions were given. 12 of them came from among the 15 Plattsburgh graduates. (Superiors said that they wrote the best, most clearly understood, .h files of the group.)

I have noticed certain concepts that seem to regularly trouble beginning students, such as parameter passing, and even the distinction between functions and procedures. I feel Ada handles both of these well, in the first case using *in*, *out*, and *in out* to describe the *abstraction* of how parameters are to be used rather than the *implementation* details of how the parameters are passed (value, reference, etc.). Also, C++ allows the programmer to call functions as procedures, ignoring any returned value (and thus disallowing the return type as a factor in disambiguating calls to overloaded functions), whereas Ada keeps the distinction clear.

### **Gender Issues**

The syntax of Ada is also more gender friendly than C++ and Java, which make heavy use of symbols and shortened terms (e.g. `int` for `integer`, `%` for `remainder/mod`, and `&&` for `and`). At SIGCSE2000, John McCormick also related the following:

At the University of Northern Iowa, a 20% increase in student retention was reported upon changing from C++ to Ada. Students were able to complete three times as many projects in the courses. But perhaps more significant is the fact that there was a 30% increase in retention among female students. While female students are often intimidated when (usually) male students with some prior knowledge of C++ ask obscure or esoteric questions in class (techies showing off?), using Ada in the first sequence usually “levels the playing field” in that few if any students come in with Ada experience.

### **When should OOP be introduced?**

There is no question that the object-oriented paradigm is an important one. This does not necessarily mean it belongs in the first course. We teach object-oriented design principles and data abstraction in our first sequence, and then expand to full OOP in the previously mentioned later courses. An informal survey of graduating seniors resulted in the overwhelming preference for Ada as the first language (not the only language!) over C++. Mike Feldman reported a similar response in the above-mentioned paper.

### **Final Remarks**

The question is not what concepts and languages to teach, but when. We must look at the undergraduate program as a whole. We cannot expect (as many of our students seem to) that the first year of the program prepares one to get a high-level summer programming job. We need to make sure that the end result after four years meets our objectives. CS-1 is the first step, but there are many more before the sum of the parts makes a coherent whole.