Programming Languages in a Liberal Arts Education

Kim Bruce
Computer Science Department
Pomona College
Claremont, CA 91711

Stephen N. Freund
Computer Science Department
Williams College
Williamstown, MA 01267

Abstract

Liberal arts curricula emphasize breadth of a student's educational experience, critical reasoning, and intellectual discourse to a greater degree than pre-professional training or engineering programs. This substantially impacts how the topic of programming languages (and computer science in general) is taught. We highlight some key aspects of teaching programming languages in a liberal arts program, and we discuss why we believe this approach prepares students for problems they will encounter throughout their careers.

Categories and Subject Descriptors K.3.2 [*Computers and Education*]: Computer and Information Science Education—Computer Science Education; D.3.3 [*Programming Languages*]: Language Constructs and Features

General Terms Languages

Keywords Programming languages curriculum

1. A Liberal Arts Education

Liberal arts colleges primarily focus on undergraduate education, and they share a strong focus on broadening the students' critical thinking, communication, and reasoning skills. At Williams College, the core goal of the institution is embodied in its mission to impart upon students the capacity

> *to explore widely and deeply, think critically, reason empirically, express clearly, and connect ideas creatively* [Wil08].

Such a mission carries a sense that the education received in college must serve the student for many years, and far beyond our predictions for what specific skills will be valuable in the future. A liberal arts education attempts to provide students with the maturity to think creatively and adapt to new situations with an "intellectual resilience that equips [them] for lifelong learning [Pom08]".

Despite some differences in the curricula among liberal arts colleges, there are several elements common to all:

- *Breadth of experience.* A key aspect of a liberal arts education is exposure to a breadth of ideas and disciplines. In some cases, students may take as many as 60–70% of their courses in areas outside of their major. By structuring a curriculum in this way, students come out well trained to see connections between related, but perhaps subtle, concepts and better able to adapt to new situations.

2008 SIGPLAN Workshop on Programming Language Curriculum, May 29–30, 2008, Cambridge, Massachusetts, USA.

In addition, it provides the opportunity for studying multiple disciplines in greater depth. For example, it is common for students to major in computer science and minor in media studies, linguistics, cognitive science, economics, or any number of other fields.

- *Reasoning skills.* Liberal arts programs de-emphasize teaching fixed knowledge and technical skills. While these items are important to students and cannot be ignored completely, liberal arts colleges believe that undergraduate courses should not be designed as professional training in preparation for a specific career. Such knowledge has a limited lifetime (of perhaps only a few short years) and may not serve students well. Moreover, it is anticipated that many students will alter their career paths, or switch careers entirely, and find themselves needing to learn new bodies of knowledge.

  A liberal arts education attempts to expose students to the fundamental principles of a discipline, problem solving and analytic skills applicable in many domains, and the critical reasoning and communication skills necessary to learn and discuss new knowledge.

  One can view this approach as "just in case" learning as opposed to "just in time" learning. Rather than learning facts and skills that have a short shelf life, student learn how to learn.

- *Intellectual discourse and communication.* Classes are typically quite small. The additional personalized attention and classroom discussion enabled by small courses is central to fostering intellectual discourse and supporting the educational goals mentioned above.

1.1 Computer Science Curricula

In general terms, computer science is the "systematic study of algorithms and data structures, specifically (1) their formal properties, (2) their mechanical and linguistic realizations, and (3) their applications" [GT86]. This definition has characterized the discipline well since first expressed twenty years ago, despite the many ways in which computer science has changed and evolved since that time.

Computer science programs at liberal arts colleges typically prioritize the study of formal properties and foundations over items 2 and 3 above. Focusing on foundations develops in students precisely those capabilities and skills at the heart of a liberal arts education. The *Liberal Arts in Computer Science Consortium* identifies three in particular [Con07]:

1. the ability to organize and synthesize ideas,
2. the ability to reason in a logical manner and solve problems, and
3. the ability to communicate ideas to others.

Thus, a graduate from a liberal arts computer science program is expected to understand the fundamental limitations of computing, know how to apply theoretical results in realistic settings, and be able to think critically and apply a variety of problem solving techniques. Moreover, a well-trained computer scientist must be able to work effectively and communicate these ideas to others. In other words, computer science programs at liberal arts colleges prepare students with "just in case" learning skills.

2. Programming Languages as a Liberal Art

An undergraduate programming languages course has long been a standard part of a liberal arts program [GT86, WS96, Con07]. The goal of this course is to give students a framework in which to understand, reason about, and adapt to new languages and programming paradigms. In particular, the course prepares students not only to understand the design and implementation of current programming languages, but also to understand how the principles behind those languages are likely to impact languages designed for future computing platforms and application domains.

To this end, the course does not explicitly aim to make students proficient in any one particular language. Rather, in keeping with the spirit of a liberal arts program, the course attempts to give students the skills to recognize the issues underlying languages and to make informed decisions when confronted with new

Topic	Class Hours
Computability	2
Syntax and Semantics, Parsing	3
Compilers, Interpreters, Virtual Machines	2
Functional Programming, Evaluation Strategies	6
Types, Polymorphism, Type Inference	5
PL Implementation: Stack, Heap, Garbage Collection, Closures	4
Control Structures, Exceptions, Continuations	3
Data Abstraction and Modularity	3
OOP: Design, Implementation, and Types	6
Concurrency, Security, and Distributed Computing	3
Domain-Specific Languages	2

Figure 1. PL Course Outline at Williams and Pomona.

languages or new problems in the future. This is not to say that programming has no role in the course. It is in fact quite important that students write enough programs in representative languages that they learn how to approach designing programs in ways best supported by the various programming paradigms. This programming experience also helps students to make informed decisions about language choice when confronted with specific problems to solve.

Figure 1 contains a rough outline of courses recently taught at Williams College and Pomona College [Fre08, Bru08] using this model. The actual topics covered are not much different than one may find in any number of courses on programming languages. The first half of the course illustrates the main ideas in Lisp (or Scheme) and ML, and the second half typically uses some subset of Smalltalk, Java, C#, and C++. Students do write small programs in these languages to reinforce basic concepts and programming paradigms, and to teach them how to quickly adapt to and use new languages in an informed manner. However, most of the class time and course work focuses on other issues:

- *Principled foundations.* Examining programming languages from first principles is key to this approach. The formal underpinnings provide a consistent framework in which to explore and discuss the material. Moreover, the theoretical aspects of this course tie together many core computer science topics, including data structure design, algorithms, computer organization, models of computation, computability, and so on. Given the limited size of a typical liberal arts major, the programming languages course may be one of only a few places where a student sees connections to so many different aspects of computer science.
- *Formal representations.* Programming languages are formal representations of data structures and algorithms for solving problems. Most computer scientists are familiar with the formal presentation of the syntax of a language via a context-free grammar, and how that connects directly to the construction of a parser. However, far fewer are aware of the formal representation of the type system, and how that can lead directly to the development of a type-checking algorithm. Finally, formal representations of the semantics of programming languages (e.g., in a modern operational semantics) lead directly to simple implementations of interpreters for a language.
- *Critical analysis of languages.* Another important element of the course is exploring the design space of programming languages from different points of view: Why was a language designed? What was its intended application? For what sorts of problems is it well-suited or ill-suited? Was it successful? How does the language compare to others in terms of expressiveness, efficiency, simplicity, and so on? These discussions force students to develop the reasoning skills necessary to adapt to and appreciate new ideas.

- *Recurring themes in language design.* The course is also intended to train students to recognize and think about recurring themes in the discipline. For example, even though only a few lectures are explicitly dedicated to concurrency, parallelism and its implications are highlighted at numerous points in the semester. This topic comes up when we discuss evaluation strategies and pure languages; higher-order functions, where we highlight their use in defining MapReduce [DG08] and other large-scale programming idioms; garbage collection; virtual machines; types, where we illustrate how to apply type inference to the problem of identifying race conditions; and so on.

  Other themes covered throughout the semester include traditional notions like abstraction, specification, usability, but also ones that have only more recently become part of the programming languages culture, such as security.

- *Reading and writing about languages.* Students read, write about, and discuss papers selected from the primary literature. The readings include research papers (on topics like garbage collection and program analysis), as well as first-hand accounts and historical perspectives on language design and evolution (as in [Str94, Ing81, GLS99]). These readings give students insights about contemporary language issues, how and why languages have been designed the way they are, and where the area may be headed.

Programming language courses serve a broad range of purposes. At one end, engineering-focused courses provide students with specific, and perhaps immediately applicable, skills related to specific languages. At the other are courses like the one we have described here. The emphasis on foundations, critical comparison, and analytic skills sets this approach apart from others.

The "liberal arts" emphasis may leave students without the same level of programming proficiency in the languages explored as students taking more programming-intensive courses. This inevitably leads to some question as to whether the missing domain-specific or language-specific knowledge will be a hindrance to our students immediately following graduation.

Certainly, potential employers or graduate schools could view the absence of specific technical knowledge as a weakness. However, we believe, and our experience has shown, that liberal arts students have a real advantage. Their broader perspective and learning skills prove to be quite valuable, and the ability to learn new knowledge and effectively communicate ideas enables them to adapt to new challenges and succeed in a wider variety of situations.

3. Summary

The basic philosophy behind the course sketched in the previous section is applicable beyond the institutions where it is currently used. It exposes students to the core programming language concepts; gives students a context in which to discuss, learn, and adapt to new languages or programming paradigms; and is effective at developing problem solving skills applicable to a wide variety of situations. We believe that designing a programming languages course to develop these skills best prepares students for the challenges that they will face in the future.

Acknowledgments

We thank Sara Owsley Sood and Duane Bailey for their valuable comments.

References

[Bru08] Kim Bruce. CSC 131: Principles of programming languages. Course website at `http://www.cs.pomona.edu/~kim`, 2008.

[Con07] Liberal Arts Computer Science Consortium. A 2007 model curriculum for a liberal arts degree in computer science. *J. Educ. Resour. Comput.*, 7(2):2, 2007.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[Fre08] Stephen Freund. CSCI 334: Principles of programming languages. Course website at `http://www.williams.edu/~freund`, 2008.

[GLS99] Jr. Guy L. Steele. Growing a language. *Higher-Order and Symbolic Computation*, 12(3):221–236, 1999.

[GT86] Norman E. Gibbs and Allen B. Tucker. A model curriculum for a liberal arts degree in computer science. *Commun. ACM*, 29(3):202–210, 1986.

[Ing81] Daniel H. H. Ingalls. Design principles behind Smalltalk. *BYTE Magazine*, August 1981.

[Pom08] The Pomona College curriculum. At `http://www.pomona.edu/ADWR/Registrar/Overview/Curriculum.shtml`, 2008.

[Str94] Bjarne Stroustrup. *The design and evolution of C++*. 1994.

[Wil08] Williams College mission statement. At `http://www.williams.edu/`, 2008.

[WS96] Henry MacKay Walker and G. Michael Schneider. A revised model curriculum for a liberal arts degree in computer science. *Commun. ACM*, 39(12):85–95, 1996.