

Objectives for the First Year

At Calvin College, our primary objective for the first year is that students leave CS-2 as competent problem-solvers. To achieve this end, we require each student to solve a sequence of increasingly difficult problems over the course of the year. Computer programming is presented as a problem-solving skill. We also spend one hour of instruction each week on a “breadth” topic to attack the misconception that computer science equals programming.

A secondary objective is that students leave CS-2 with competence in a modern problem-solving methodology, namely object-oriented design (OOD). To achieve this, they must understand classes, inheritance, and polymorphism, as well as the ‘traditional’ programming topics. In CS-1, our students learn to design and build classes, and see (but get little practice using) inheritance and polymorphism. In CS-2, our students get further practice designing and building container classes, and learn to incorporate inheritance and polymorphism into their designs. At present, C++ is the language used in both courses.

Problem-Solving Methodology

In our experience, novice programmers are unable to make use of OOD due to their inexperience. As a remedy, we have developed **object-centered design** (OCD) [1], a graduated five-phase design methodology that culminates in OOD. OCD has worked very well for us: it has virtually eliminated the “I don’t know where to start” syndrome that afflicts so many novice programmers. It helps that our textbooks for CS1 and 2 ([2][3]) reinforce this methodology.

Our CS-1 course focuses on the first three phases of OCD. In the first phase, students describe the *behavior* of their program (what they want it to do), identify the *objects* in their description (its nouns), identify the *operations* in their description (its verbs), and organize the objects and operations into an *algorithm*. In the second phase, students extend the first phase by defining functions for operations that are not predefined. In the third phase, students extend the second phase by defining classes for objects whose types are not predefined. The problems we ask students to solve are carefully chosen to be appropriate for a given stage. The traditional topics of CS-1 (expressions, control structures, files, arrays, etc.) are covered in the context of the OCD

framework. Inheritance and polymorphism are introduced in the last two weeks of the course, but primarily as a lead-in to CS-2.

CS-2 focuses on the last two phases of OCD, which emphasize analysis. In the fourth phase, students extend the third phase with an analysis step of organizing related classes into hierarchies, so as to eliminate redundant coding via inheritance. In the fifth and final phase, students extend the fourth by making use of polymorphism where appropriate in their class hierarchies, resulting in OOD. Students receive extensive experience with the traditional topics of CS-2 (pointers, vectors, linked lists, stacks, queues, trees) but the focus is on their use in software design, rather than as isolated topics. Templates and the C++ standard template library are also studied.

The Future

We are currently discussing switching the language of instruction from C++ to Java in CS-1. While we have been reasonably happy with C++ in CS-1, these arguments have been advanced for how Java would “add value” to CS-1 compared to C++:

- Using Java in CS-1 would have the benefit of providing students with more practice building classes (i.e., OCD phase 3 would become phase 1) and so produce students who are better prepared for our CS-2 course.
- Java might allow us to teach GUI programming in CS-1, which students would find more interesting and motivating than their programs’ current text-based interfaces.
- Because advanced topics like multithreading and network programming are relatively simple to use in Java, “breadth” sessions on these topics could be made more accessible to students than is the case with C++.

Regardless of what we do in CS-1, we see no advantages switching from C++ in CS-2. In fact, we view some of the supposed “advantages” of Java (e.g., garbage collection, hidden pointers, lack of parameterized classes, etc.) to be distinct drawbacks to learning the discipline of building container classes using dynamic memory allocation.

References

- [1] J. Adams, “Object-Centered Design: A Five-Phase Introduction to Object-Oriented Programming in CS1-2”, *Twenty-Seventh SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, Pennsylvania, February 1996.
- [2] J. Adams, S. Leestma, L. Nyhoff, “C++ *An Introduction to Computing*” (2nd Ed), Prentice-Hall, 1997.
- [3] L. Nyhoff, “C++ *An Introduction to Data Structures*”, Prentice-Hall, 1998.