Web Labs for the Standard Template Library and the Java Generic Library in a CS2 Course

William J. Collins
Computer Science Department
Lafayette College
Easton, PA 18042
610.330.5228
collinsw@lafayette.edu

Yi Sun
Computer Science Department
Stanford University
Stanford, CA 94305
650.497.6221
yisun@cs.stanford.edu

1. Abstract

This paper describes a suite of laboratory experiments for a CS2 course. The main thrust of the experiments is to promote an understanding of a container-class library: the Standard Template Library for C++ or the Java Generic Library for Java. All of the experiments are stored on the World Wide Web, and students have considerable latitude on when each experiment must be completed.

2. Introduction

The CS2 course has traditionally focused on data structures, a term which, in an object-oriented context, refers to container classes. In the CS2 course at Lafayette College, the lecture section introduces such container classes as linked lists, stacks, priority queues, red-black trees and hash tables.

In the laboratory section, students perform experiments related to each container class. In each experiment, there are subsections for Observing, Hypothesizing, Testing and Concluding. *Computing as a Discipline* (Denning [1989]) promotes the use of such labs to help students "learn to distinguish careful experiments from casual observations". Similarly, *Computing Curricula 1991* (Tucker [1991]) recommends laboratory experiences involving hypothesis formation and testing because they "increase student problem solving ability, analytical skill, and professional judgment".

One of the goals of the course is for students to carefully study code written by professionals. After all, we expect that some of these students will eventually become professionals and write highly efficient (but readable) code. A standard library of container classes is utilized instead of a "home-grown" version of these classes. The code in these classes is somewhat inscrutable, and a laboratory environment is an ideal setting for line-by-line analysis. A standard library has the added advantage of portability, so students can be confident that their applications will still work in other locales.

SIGCSE '99 3/99 New Orleans, LA, USA

The Standard Template Library (see Musser [1996]) is part of ANSI Standard C++. There are three major components: container classes, generic algorithms that can be called by container objects, and iterators. Iterators provide a consistent interface to the container classes so that the generic algorithms can work without relying on details of those classes.

The Java Generic Library, from Objectspace, Inc., is the Java analog of the Standard Template Library. The language feature that most clearly distinguishes the two libraries is the template facility: ubiquitous in the Standard Template Library but not available -- at least not yet -- in Java. In the Java Generic Library, the type of each item in a container is Object, which a user can typecast to a type suitable for the current application.

All of the laboratory materials are stored on the World Wide Web. The pedagogical advantages of this medium have been well documented (Carlson [1996], Hitz [1997], Paxton [1996]). A further benefit of the World Wide Web is that its hyperlink, mail, and security features support the separation of the labs into separate stages for observation, hypothesis formation, testing and conclusions.

2. Course Outline

The CS2 course at Lafayette College presents a study of container classes. This is a four-credit course, with 150 minutes of lecture and two 75-minute labs per week. In the lectures, each container class is described, its design is outlined and analyzed, and then an application that uses the container class is developed. Most of the students in the

course took an object-oriented CS1 course with Java, so they will use Java in the CS2 course. But students in Electrical and Computer Engineering are required to use C++, and students with Advanced Placement credit in computer science may prefer C++ to Java.

Much of the lecture material is language-independent, with illustrations of how methods work -- such as inserting into a doubly linked list -- rather than on language details. Occasionally, the differences between Java and C++ warrant discussion. These contrasts highlight significant language-design issues even for students who are familiar with only one of the languages.

For example, Java prizes security over efficiency, so the index checking in the Quick Sort generic algorithm contributes to that method's (two orders of magnitude) slower run-time than the corresponding C++ version. Another factor impeding Java's speed is that -- as you would expect in an object-oriented language -- methods are virtual by default, so inlining is not possible. C++, a hybrid language, facilitates inlining: any method defined within a class body is automatically inlined, and a function outside of a class body can be inlined with the **inline** keyword. This gives C++ programmers the semantics of a function call without the save/return overhead.

There are two sets of lecture notes, *Data Structures and the Standard Template Library* and *Data Structures and the Java Generic Library*. Both sets can be downloaded from the course home page. These notes cover the usual data-structures/software-engineering material: recursion, analysis of algorithms, formal verification and so on. Material on object-orientation is also included: inheritance, polymorphism and the major container classes.

The lab materials must cover the language features in sufficient detail to allow students to understand the library code and to complete the assigned programming projects -- such as enhancing a text editor or spell checker, or developing a condition evaluator.

The course grade is based on 1000 points, distributed as follows: 150 points for lab assignments, 350 points for programming projects, 200 points for the midterm exam, and 300 points for the final exam.

3. The Laboratory Component

The laboratory consists of 24 Pentium-chip computers running Borland C++ 5.02 and Cafe (from Symantec, Inc.) under Windows 95. The computers are part of the campus network, which has World Wide Web access through Netscape.

There are 28 lab periods, and each lab period is 75 minutes long. The labs are *closed* in the sense that there is a fixed lab period by the end of which the lab experiment must be completed and during which the instructor is present in the lab. But students need not attend the lab period as long as they e-mail their conclusions by the end of the lab period. All students must e-mail their hypotheses *before* the beginning of the lab period.

The lab assignments, on the World Wide Web, are accessible through the following cover page Uniform Resource Locator (URL):

http://www.cs.lafayette.edu/cgi-bin/cs103/backupcpp

While working on a lab assignment, a student will go back and forth between a Web browser such as Netscape, the C++ or Java Integrated Development Environment, and a mail facility such as PINE.

For the first ten labs, the focus is on language features that are not necessarily covered in a CS1 course. For C++, for example, there are labs on creating overloaded operators, iterators and templates. For Java, some of the early labs are on interfaces, exception handling and synchronization. For the remaining 18 labs, the emphasis is on the standard library, especially its container classes.

The labs are written in HTML (Hyper-Text Markup Language). There are two advantages to this. Because Internet access is available throughout the college, labs written entirely in HTML will give students the opportunity to do the labs anytime and, with a laptop, anywhere. Secondly, the hyperlinks in HTML provide an easy way to direct the flow of the lab: students are led through the process of observation, hypothesis formation, testing and conclusion.

Each laboratory experiment consists of four sections: Observation, Hypothesis Formation, Testing, and Conclusions. We now describe each section in detail.

3.1 Observation Section

In this section, there is a discussion of the lab topic with hyperlinks to a glossary, code examples and, occasionally, images. After about 50 lines of hypertext, there is a simple, multiple-choice quiz. A student who clicks on a wrong answer is sent back to review the previous section -- with a hint explaining why the student's choice was incorrect. A click on the right answer advances the student to the next section.

For example, The Java version of Lab 27 tests the run times for various sort methods: BinarySearchTree Sort, RedBlackTree Sort, Heap Sort and Quick Sort (the C++ version also tests Merge Sort, the only sort method provided for the *list* class in the Standard Template Library). In the lectures before this lab, students had been exposed to these sorts and a Big-O analysis of their average and worst times. An early quiz from that lab deals with the run times for

sorting random integers with RedBlackTree Sort:

Quick Quiz: When the project (SortTimer1.java) was run -- on a 200-megahertz Pentium -- with 10,000 random integers, the run time was 1.59 seconds. Estimate the run time for 20000 random integers. (Hint: See Exercise 9.5a.)

a. 3.18 seconds

b. 3.40 seconds

c. 6.12 seconds

d. 9.68 seconds

(Exercise 9.5a is given in the Appendix). In the actual lab, the choices are hyperlinked. In this paper, underlining is used to indicate hyperlinks. By choosing b, the student gets to advance to the next page of the lab. Any other choice will include a hint and a request to review this part of the lab. Such quizzes ensure that the students will be actively involved in the material. After, on average, two such quizzes, the Hypothesis Formation section is entered.

3.2 Hypothesis Formation Section

Here the student conjectures "what would happen if ..." or "what needs to be added in order to ...". For example, here is the Hypothesis section of the Sorting Times Lab :

HYPOTHESIZE:

Modify SortTimer1.java to determine the time for BinarySearchTree Sort. Which do you think will be faster in the average case (that is, for sorting random integers), BinarySearchTree Sort or RedBlackTree Sort? Why? Which do you think will be faster in the worst case? How would you get the worst case?

Use Exercise 9.4 to figure out how to arrange n items to get the worst time for Quick Sort. Write the code to store the n items in an Array called "array".

Hypothesis

(Exercise 9.4 is given in the Appendix) A crucial feature of the Hypothesis Formation section is that students do not yet have easy access to all of the code they will need to test their hypotheses. This is an anti-hacking measure.

The hyperlink on the Hypothesis Formation page leads to a form where students can fill in their hypotheses. Hypothesis submission is accomplished through a Common Gateway Interface (CGI) script. When the student clicks on the "SEND" button, two mail messages are simultaneously generated:

1. The contents of the Hypothesis window are e-mailed to the instructor. This is achieved through HTML's form handling capability and a shell command "sendmail". The mail must be sent by the student **before** the start of the lab period. This portion of the lab is graded -- 40% of total lab grade -- on the reasonableness of the hypotheses.

2. The password for this lab assignment is automatically e-mailed to the student. The student uses this password to access the Testing section. For example, the password for Lab 3 is "cat", so the URL to access the Testing section for Lab 3 is

www.cs.lafayette.edu/~collinsw/cs103/labs/cpp/l27/cat.html

3.3 Testing Section

The student is given the details of an experiment that will test the hypotheses. This section includes hyperlinks to relevant files, sometimes with missing parts. The student transfers these files to disk. Within the Borland C++ or Cafe environment, the student modifies the given files, adds new files if necessary, creates a project, and runs that project with supplied or generated input. For example, here is the testing section of the Sorting Times Lab:

TEST:

Run your modified-to-get-BinarySearchTree Sort for SortTimer1.java with 10,000 random integers and with 20,000 random integers. You will need to add BinSearchTree.java, BinSearchTreeIterator.java, and Node.java to your project. Then run the project for RedBlackTree Sort for 10,000 and 20,000 random integers. For the worst times for BinarySearchTree Sort and RedBlackTree Sort, run the project for 1,000 items and for 2,000 items.

Run your modified-to-get-worst-case version of SortTimer3.java (Quick Sort) for one thousand items and for two thousand items. If your time for two thousand items is not about four times as large as your time for one thousand items, your code does not produce the worst case. Keep trying!

On completion of the testing section, the student goes back to Netscape and enters the Conclusions section.

3.4 Conclusions Section

In this section, the student fills in a window with a brief report on the results of the experiment and what the student learned. Especially important are explanations of any discrepancies between what the student hypothesized and what the student discovered during testing. For example, here is the Conclusions section of the Sorting Times Lab:

CONCLUDE:

Include your BinarySearchTree modification to SortTimer1.java and the run times you got for 10,000 random integers and for 20,000 integers. What were the corresponding times for RedBlackTree Sort? What's going on here? Shouldn't RedBlackTree Sort be **faster** than BinarySearchTree Sort? When would RedBlackTree Sort be preferable to BinarySearchTree Sort?

Include your code for getting the worst case with Quick Sort and the run times you got for 1,000 and 2,000 integers.

Quick Sort is generally considered the fastest sort algorithm on average. Why is that not true for the Java Generic Library's version?

Explain any discrepancies between your hypotheses and your results in the lab.

What part of this lab gave you the most trouble?

Conclude

The Conclude hyperlink leads to a form. When the student fills in that form and clicks on Send, the contents of the form are e-mailed to the instructor. The conclusions form is processed in the same way as the hypothesis-formation form. A CGI script e-mails to the instructor the conclusions entered by the student. This portion of the lab grade -- 60% of total lab grade -- is based on the extent to which the student succeeded in testing the original (or modified) hypotheses and in answering the questions asked.

These suites of labs also include other aspects of an ordinary lab class. Students are able to receive feedback -- grades and professor's comments -- through the World Wide Web as well. This is done by letting each student have an alias, which is the name of a file on the Web server. The instructor posts the grades and comments for each lab on each individual's file. The students can then pull up the file on Netscape and view their own grades and the comments.

4. Conclusions of this Paper

By working from a standard library of container classes, students can be walked through code written by professionals. All of the source code in the Standard Template Library and Java Generic Library is available, so the labs give students practice in deciphering such code as the following (from the *get_node* method in the Standard Template Library's *list* class):

```
return free_list ?
        (free_list = (link_type)( free_list->next), tmp)
        : (next_avail==last ?
                (add_new_buffer(), next_avail++)
                : next_avail++);
```

This statement is not too difficult to unravel once it is pointed out that both the conditional and comma operators are involved. This sort of **code density** is vastly different from the **simplicity** that students usually encounter in lectures and textbooks.

The container classes and generic algorithms of these standard libraries constitute building blocks for later courses. This advantage is augmented by another feature of these libraries: portability. Both libraries are widely available on a variety of platforms, so students need not rely on the instructor's library. Often, an instructor's library is only partially tested, and that testing is done on a single platform.

There is also an inter-language transference. The Java Generic Library was developed from the Standard Template Library, and so both libraries have the same components: container classes, generic algorithms, iterators, function objects and adaptors. There are a few key differences; for example, the Java Generic Library relies more heavily on function objects (because operator overloading is not allowed) and exception handling. But basically, understanding one of these standard libraries makes it substantially easier to understand the other. This transference is especially valuable at Lafayette College, where Java is used in the CS1 and CS2 courses, and C++ is used in subsequent courses.

Students learned -- possibly through the osmosis of 28 labs -- a systematic approach to experimentation. Hacking was discouraged by the separation of the Hypothesis Formation section and the Testing section: Until the hypotheses were e-mailed (for grading) to the instructor, students could not easily access the code needed for testing. Also, the quizzes, hypotheses and conclusions ensured that the active learning went beyond coding.

Each lab took about three hours to complete. According to the written evaluations, some students thought this was too much time spent on labs. But the quality of the student projects -- which the labs were preparation for -- was exceptionally high. Also, the lab schedule was advantageous for the more capable students: they usually completed the entire lab assignment before the lab period, so they were able to skip the lab period. And weaker students profited because the instructor could devote more time to each of them during the lab period.

One drawback to using the Web is that the amount of time to construct lab assignments on the Web is quite a bit more than for other lab assignments. The student does most of the work out of the presence of the instructor, so the files, hyperlinks and passwords must be carefully tested beforehand. Otherwise, students who embark on a lab at 2

a.m. will send flame mail to the instructor.

The most positive aspect of using a standard library seems to have been on students' attitudes toward the course. This is difficult to analyze objectively, and it is premature to draw long-term conclusions based on only two semesters. But perhaps because they were able to understand code written by professionals, students felt more like professionals themselves. Whatever the reason, enthusiasm for the course has far surpassed that of previous semesters.

5. References

[1] Budd, Timothy, *Data Structures in C++ Using the Standard Template Library*, Addison-Wesley, 1998.

[2] Carlson, G.M., Guzdial, M., Kehoe, C., Shah, V., Stasko, J., "WWW Interactive Learning Environments for Computer Science Education", *SIGCSE Bulletin* 28, 1 (March 1996), pages 290-294.

[3] Denning, P.J., Comer, D.E., Gries, D., Mulder, M.C., Tucker, A.B., and Young, P.R., "Computing as a Discipline", *Communications of the ACM 32*, 1 (January 1989), pages 9-23.

[4] Hitz, M. and Kogeler, S., "Teaching C++ on the WWW", *SIGCSE Bulletin* 29, 3 (September 1997), pages 11-13.

[5] Paxton, J.T., "Webucation: Using the Web as a Classroom Tool", *SIGCSE Bulletin* 28, 1 (March 1996), pages 285-289.

[6] Musser, D.R. and Saini, A., *STL Tutorial and Reference Guide*, Addison-Wesley, 1996.

[7] Nelson, Mark, C++ Programmer's Guide to the Standard Template Library, IDG Books Worldwide, Inc., 1995.

[8] Tucker, A.B. (Editor) et al, *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force*, ACM Press.

6. Appendix

Exercise 9.4. Develop an arrangement of the integers 0...n-1 which will require the worstTime (n) for Quick Sort. Assume that the pivot is chosen as the median of the first, middle and last items.

Hint: In order to get the worst time, each partition should produce a subsection with just one item, either the smallest or largest item in the segment. One way to get this is to put the items in order except that the middle two items will be the smallest and largest items. For example, if the numbers are 0...9, we start with

1, 2, 3, 4, 0, 9, 5, 6, 7, 8

Note that when each partition is performed, the pivot is either the next-to-smallest item or the next-to-largest item, so each partition will produce a subsection of size 1.

Exercise 9.5a. Suppose we have a sort algorithm whose averageTime (n) is O (n log n). For example, any of the fast sorts in this chapter would qualify as such an algorithm. Let runTime (n) represent the time, in seconds, for the implementation of the algorithm to sort n random integers. Then we can write:

runTime (n) ~ k (c) * n * $\log_c$n seconds,

where c is a an integer variable and k is a function whose value depends on c. Show that runTime (cn) ~ runTime (n) * (c + c/$\log_c$n).