Third Edition of a Classic Disappoints

reviewed for SIGOOT by Conrad Weisert, October, 1998

Stanley B. Lippman & Josée Lajoie: *C++ Primer*, (3rd. Edition) 1998, Addison-Wesley, ISBN 0-201-82470-1

Earlier editions of Lippman's C++ Primer stood among the best of the hundreds of treatments of C++/OOP. I've used the second edition as a course text and have enthusiastically recommended it to colleagues at all levels. Sadly, the new edition no longer commands such respect.

I recommend the new book as a third or fourth C++ reference in a practicing professional's library. There's a wealth of information and some good advice here. Lippman knows in minute detail many subtleties of this vast language/library complex, and the experienced reader will undoubtedly learn a few new things, as I did. For those with less experience the book is worse than useless; it may well discourage prospective converts to C++.

Not a primer

What this book is definitely *not* is a primer, i.e. a suitable *first* text for anyone just coming to C++, whether through self-study or a formal course. The strange sequence of topics, always tricky in C++, will lead to a hopeless sense of vague confusion on the part of the inexperienced reader. Full explanations of crucial OOP concepts like *inheritance* are deferred for over 800 pages, while early case-study chapters bombard the reader with unexplained arcane or advanced issues.

STL integration

Parts of the C++ standard template library (STL) have become part of everyday repertoire of many practicing C++ programmers, and any ambitious C++ programmer must confront them sooner or later. But the STL is huge. Many of its concepts are hard to grasp at first, even for knowledgeable C++ programmers.

Lippman & Lajoie have chosen to integrate certain STL components into their introductory presentation of C++ and OOP, an ill-advised choice. The student needs some time between mastering both the language and the OOP paradigm and tackling the subtle complexities of the STL. Excellent books that focus on the STL (see January, 1997 SIGOOT Newsletter) can quickly orient the reader who has been using C++ effectively for at least a few months.

Furthermore, Lippman & Lajoie's STL introduction is anything but systematic. They *sneak* certain techniques in early, before laying any foundation. Although function templates aren't fully explained until page 489 and class templates not until page 811, various template instantiations appear throughout the introductory material and there's a steady barrage of specific containers and generic algorithms long before the reader has any context or framework to put them in. Many of the specific container or generic algorithm presentations are mere rote tutorials: Follow this model, even if you don't understand it yet, and you'll get results. Does a C++ beginner really need to manipulate Multimaps and Priority Queues before he or she has a clue about how to represent and manipulate amounts of money, dates, or other data common in either business or scientific applications, not to mention how to define new classes?

Beginnner's gaffes and missing topics

While most of the explanations and examples are correct and some are well-motivated, we find a few surprisingly naive ones, more characteristic of students' homework than a scholarly textbook. Although these problems are minor compared with the pedagogical difficulties, they may well undermine the student's respect for the authors.

Boolean confusion (p.159): In explaining the conditional operator, the authors give this example:

```
bool is_equal = (!strcmp(str1,str2) ?
                    true : false;
```

explaining that it's more compact than:

```
bool is_equal = false;
if (!strcmp(str1,str2))
     is_equal = true;
```

Since any experienced programmer or advanced student knows that

```
boolean_expression ? true : false
```

is redundant, having the same value, by definition, as

```
boolean_expression
```

we wonder how that slipped through, along with scattered similar examples. That example also calls attention to a major omission. You can search the index all day looking for **`strcmp`**, a standard C-library function. But that function, along with its **`<string.h>`** companions, meticulously explained in the second edition, have vanished, except in more examples (Did the authors forget that they had purged the explanations of those functions?) and an unindexed and incomplete summary on page 93. We concede that those C library routines were crude and error-prone, but they're part of the C++ legacy. Learning about them gives the student an appreciation of the need for a C++ string class as well as tools for building one. Nearly all other introductory C++ books retain an explanation of them.

Memory misunderstandings (pp. 27-28): In a work that includes so many detailed explanations of C++ subtleties, it's jarring to encounter a confused and just plain wrong explanation of memory allocation. The authors apparently believe that "dynamic allocation" is synonymous with controlled allocation using **new** and **delete** with a pointer (so-called "heap allocation"). They then apply the term "static" to denote everything else, including not only bona fide static storage allocated at compile time, but also *automatic* storage ("stack allocation), the C/C++ default. The latter is, of course, dynamic, occurring at run time upon entry to and exit from a block.

Recursion abuse (p. 362): "Computing the factorial of a number lends itself to . . . a recursive function."

Yes, that example appears in other books, especially older ones, but surely everyone now agrees that a recursive factorial routine is a *ridiculous* implementation. Such a claim only conveys to the reader either the mistaken notion that recursion isn't useful for everyday programming situations or that the authors are out of touch with real, practical applications programming.