Review

C++ programming style

Cargill T., Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, 1992. Type: Book (9780201563658)

Date Reviewed: Jul 1 1995

Full Text

Comparative Review

When asked what is the best C++ textbook, the answer has to be “It depends.” This review is an attempt to answer the subsidiary question “Depends on what?” and to give a comparative breakdown of the qualities of ten books. The structure developed in this review could be applied to any other book on the subject, thus enabling the reader to perform additional evaluations of books not reviewed here.

Table 1 considers initial criteria for comparing the books under consideration. Let us first consider the simple parameters. In date of publication, the books range from 1990 to 1994. The authors are from the United States, with one exception (Winder). Interestingly, only three of the authors--Wang, Winder, and Budd--are from universities. Two--Lippman and Coplien--are at AT&T Bell Laboratories (the home of C) and the others are with software firms or are consultants. This unusual variety of affiliations is an indication of the importance of C++ in the industrial arena.

The first definitive criterion is the prior knowledge each book requires. All the textbooks assume that readers have some programming experience. Lippman, Wang, and Winder carry on from any language, for example Pascal. The bulk of the books--Budd, Gorlen, Model, Ranade, and Smith--assume a knowledge of C. The last two, Cargill and Coplien, are books for those who already know C++ and wish to improve their skills. Interestingly, several books do not cover all of C++, and this could be an important factor influencing one’s choice of a book. On another level, two of the books, Budd and Model, specifically cover the traditional algorithms and data structures syllabus in addition to C++. Finally, C++ is a language for object-oriented programming, and the prominence given to this paradigm could be a determining criterion. All of these factors are summarized in Table 1.

Algorithms and Data Structures: Budd and Model

At this point, we can make a decision. If we want an algorithms and data structures book, then we must look at Budd or Model. Both books state that C is a prerequisite, and are firm about this. In fact, Budd’s first piece of code on page 33 is a class with no explanation of the syntax involved. Model is better, and has an introductory chapter going through the basics of C++ (which includes C).

Budd is a good text for a data structures course, and has a modern approach to

C++ style. All the data structures are presented as template classes, and the emphasis is on sound, standard programming practice. Model, on the other hand, presents a completely novel view of programming C++ without classes, making use of *struct* instead. To give an idea of the effect of this decision, consider the comparative code in Figures 1 and 2 of a stack in the two books.

```
//

// class stack

// abstract class--simply defines protocol for stack

//operations

//

template < class T > class stack {

public:

virtual voiddeleteAllValues()=0;

virtual intisEmpty() const=0;

virtual Tpop()=0;

virtual voidpush(T value)=0;

virtual Ttop() const=0;

};
```

FIGURE 1: Budd's version of a stack (p. 238)

```
template < class elt > struct stack

{

public

// Initialize/finalize

stack(int size = 100);

~stack();

// Access/Modify

void push(elt);

stack(elt)& operator+=(elt);

elt pop();

elt topelt();

void clear();

};
```

FIGURE 2: Model's version of a stack (p. 150)

Model justifies his use of *struct* on the basis that there is no real need to introduce a new concept to students who are already comfortable with C. However, since templates require the word class rather than *struct* as a parameter, we just have to think of this use of the term to mean the same as "type."

Budd's code is cleaner and more natural, and he has made an effort to enhance readability through good layout. Model, on the other hand, has more in the way of imaginative examples. The stack is illustrated by means of a program to read in stacked references to scenes in Shakespeare, rather than the usual calculator example found in Budd. Model also manages to cover more actual C++ syntax than Budd does.

Neither of the books is mathematically demanding, and the analysis of the algorithms in terms of big-O notation is not central to the discussion. Sections at the end of each chapter of Budd's book compare the efficiency of various implementations. Model tackles the issue mainly with respect to sorting.

One of the striking differences between the books is Model's steadfast attitude against object-oriented programming. He simply does not believe in it. Thus, he presents no inheritance or polymorphism. Budd takes the safer route, presenting all abstractions as potentially polymorphic, but the issue is not developed throughout the book, and no real example of inheritance is presented.

Budd has reasonable exercises at the end of each chapter, but Model's are more extensive and show a greater variety. Table 2 summarizes the differences in resources between the two books, including the compilers on which the code has been tested.

Table 1: Knowledge assumed; coverage of topics

	Knowledge assumed	Coverage of C++	Coverage of algorithms and data structures	Coverage of OOPS
Budd	C	little	complete	little
Model	C	some	complete	none
Lippman	none	complete	little	some
Winder	any language	complete	extensive	extensive
Wang	any language	complete	little	extensive
Gorlen et al.	C	little	little	extensive
Ranade & Zamir	C	extensive	none	some
Smith	C	extensive	none	some
Cargill	C++	advanced	little	extensive
Coplien	C++	advanced	some	extensive

The choice between these two books is not an enviable one. Budd is probably the sounder and safer text, but the practical sessions for a course would have to be augmented by material from other books such as Model's. Finally, since both books assume a knowledge of C and do not cover all of C++, one might have to

consider Winder as an alternative.

C++ without C: Lippman, Winder, and Wang

We now look at Lippman, Winder, and Wang, which essentially cover advanced and object-oriented programming while introducing C++ as a language in its own right. No knowledge of C is assumed, and all three books seem to cover all of the C++ syntax. The choice here is far more difficult, as it has to be made on how well the book puts across such concepts as inheritance and polymorphism, and on the depth of C++ detail that is covered.

Lippman's book is quite old even in its second edition, and it follows a reference manual ordering of C++ topics from types to statements to classes to inheritance, even putting I/O right at the end. Many little examples illustrate small points, but the book includes no big examples that show the power of C++ for programming-in-the-large. The discussion of object-oriented programming and design is left to the end of the book. There are a few diagrams and tables.

Winder, as his book's title suggests, does not set out to provide an algorithm and data structures book, but the coverage is such that it would almost qualify under this heading. In addition to the normal data structures, he develops substantial classes based on a wide range of easy-to-understand objects such as complex numbers, dates, people, and vehicles, and uses these in subsequent programs. He makes use of inheritance and polymorphism and looks at code reuse.

A notable feature of Winder is that he does not underestimate the difficulties of C++ and goes to pains to explain the problems of real C++ compilation, for example Gnu++ v2.2.2 requiring static members to be defined within class definitions to be compilable, thus making simple classes quite unreadable. This kind of detail is valuable for programmers, but may well throw the student of data structures off balance. I would value Winder's book more for a course in advanced programming, for students who perhaps do not have a traditional computer science background and will get the data structures exposure for free. If Winder is chosen for a data structures course per se, then the algorithm material will have to be augmented from elsewhere and the later data structures, such as graphs, added.

Wang's book is a brand new offering and one of the books that impressed me the most. Like Lippman, it goes through the syntax of C++, but the tempo is quicker and more suited to a second-level programmer. The opening basics are over by page 160, when classes appear. Thereafter, we find I/O, libraries, inheritance, overloading, templates, and a thorough approach to object-oriented programming and design. A practical chapter on compiling programs rounds the book out. To get an idea of the kind of example used, consider the sequence of dates derived from an ordered sequential class, shown in Figure 3.

```
//////File Date_OS.h//////

#include “Date.h”

#include “OrderedSeq.h”

class Date_OS : public OrderedSeq // derive from abstract // base

{ public:
```

```
Date_OS() : len() { }// default constructor

Uintlength() { return(len); } // current length of seq

Date*operator[](Uint i);// overloading []

voidremove(Uint i);// remove by index

protected:

intappend(void* date);// add any at end, -1 failed

private:

voidswap(Uint i, Uint j);// interchange elements

intcmp(Uint i, Uint j);// compare elements

intcmp(void* date, Uint j); // compare key to // element

enum{Max=256};// maximum size

Date*dates[Max];// pointer array

Uintlen;// total no. of dates

};
```

FIGURE 3: Inheritance from Wang (p. 317)

Wang does not cover data structures, but nevertheless derives classes for some common ones and some unusual ones (ordered sequential, dates, times, and polynomials). The approach is less intense than Winder's, and is more likely to be understandable at the second level. A large example of a windows calculator brings together all the ideas of class derivation, multiple inheritance, and polymorphism and shows the power of C++ well. The technical appendices are worthwhile, and include information about compiling under UNIX (Gnu C++ or USL CC) or on a PC (Borland or Microsoft).

Table 2 once again shows the resources available in and for the three books. Overall, Wang is the most comprehensive. To choose in this section, I would exclude Lippman and go for Winder or Wang. Winder could, at a push, double as a data structures book, but care must be taken in using it with novice programmers. Wang in all respects is an excellent book, but does not have comprehensive coverage of data structures. Incidentally, both Wang and Winder include in their index the notorious C++ operator tokens, an addition that will be invaluable to beginners.

C++ after C: Gorlen, Orlow, and Plexico; Ranade and Zamir; and Smith

In this category, we have Gorlen et al., Ranade and Zamir, and Smith. The first question that must be asked is how serious the authors are about the C knowledge they require. They all use comments such as "as in C," but if one scans the first few chapters of each book, it is clear that a competent programmer familiar with some other language could catch on quite quickly. What one does not want to do is to teach C first just to get into any of these books. First, this would

waste time, and second, several authors support the notion that C++ is a language in its own right and not just an extension of C. Better programming habits can be developed by starting with C++ directly. In summary, therefore, I would not regard the required C knowledge of these books as a stumbling block to their adoption in a second course for, say, Pascal, Ada, or Smalltalk programmers. Those from the functional or declarative schools may have more difficulty, however.

A surprising feature, and a big disadvantage of all three books, is that they do not cover the whole C++ language. Specifically, templates are ignored. Since C++ is intended for advanced programming, it seems impossible that one could get along without such a feature. Why use the language otherwise? The reason for the omission is probably related to the books' ages--1990, 1992, and 1990. Templates were not supported by most compilers until late in 1992.

A second disadvantage is that these books do not give exercises. One must assume that the writers are targeting the workplace rather than the classroom. This is an important factor for lecturers to consider. Nevertheless, let us consider what these books have to offer. Gorlen presents an intensive study of C++ through the development of a real class library used by the National Institutes of Health. The C++ is clear and well presented, and the book includes many complete examples that illustrate how the classes actually work. For example, the three classes "Patients," "Set," and "SortedCltn" (a sorted array of object pointers) are combined to create a small database, answer queries, and produce lists of patients. Sample runs of these programs are always given, adding to their authenticity. The final example of a traffic simulation draws together the full range of object-oriented programming techniques that have been covered throughout the book.

Ranade's book is in content a much slimmer offering. In the early part of the book, many small examples illustrate syntax, but the text is terse, lacking the richness of the others previously discussed. Ranade even claims that each chapter can be read in an hour, and so it will only take 24 hours to finish the book. This may be an advantage. Some readers like a minimum of words and a maximum of examples. The two programs developed at the end of the book are good and illustrate C++ inheritance.

Smith centers around the development of a large windowing and editing system, including detailed interfacing to screens, windows, and keyboards via the Zortech compiler running on PCs under MS-DOS. Smith acknowledges that his code is not completely portable, but it is reasonably so. Overall, this book would be suitable for a systems programming course, where C and C++ are incidental, and the main objective is to show how good-quality, low-level software can be built with a modern high-level language. The resulting systems are given in both C and C++, which is overkill for our purposes, but perhaps makes the book more versatile. The code is clear and readable, and the proper use of abstraction and object-oriented programming is stressed throughout. Table 2 summarizes the differences in resources among the three books.

Advanced C++: Cargill and Coplien

Table 2: Resources available

	Examples	Exercises	C++ source	Compilers
Budd	Fair	Some	Via ftp	AT&T, GNU, Borland Turbo
Model	Good	Many	Disk with book	Borland V. 3, Digital (Ultrix, UNIX)
Lippman	Mostly small	Some	Not available	AT&T Rel. 3.0
Winder	Good	Few	Via ftp	Borland V. 3.1
Wang	Good	Some	Disk to order	AT&T, GNU, Microsoft, Borland
Gorlen et al.	Good	None	Via ftp	AT&T Rel. 2.0
Ranade & Zamir	Fair	None	Disk with book	Borland Turbo
Smith	Fair	None	Disk to order	Zortech V. 2.0
Cargill	Good	Few	Not available	Not specified
Coplien	Good	Many	Via ftp	AT&T Rel. 4, GNU V. 1.39.0, Zortech V. 2.0

The last two books assume C++ knowledge and aim to take it further. As their titles suggest, they are concerned with style. For an accomplished programmer, they are a joy to read.

Cargill's book is based on articles he wrote for a journal, and each of the nine chapters tackles a specific topic relating to good programming in C++ and good object-oriented programming in general. For example, chapter 3 looks at unnecessary inheritance. It starts with a Stack class, from which IntStack and CharStack are inherited, and then shows how using public inheritance in this way creates a dangerous hole in the encapsulation of the stacks. He presents several better solutions, including using templates. In fact, the whole book is concerned with critically assessing and improving programs written by others and gleaned from textbooks, magazine articles, and tutorials on C++.

The code in Cargill is always well laid out and easy to read. The examples in the first seven chapters center around classes, and then chapter 8 develops a case study that is intended to show the power of C++. Unfortunately, the topic of the case study is not the best, as a finite state machine simulator is really a simple program. Nevertheless, some good points of abstraction and inheritance are reinforced.

Coplien is genuinely an advanced C++ book and could be a natural extension to one of the books in the previous groups. It hammers home the use of classes and templates, and devotes a good deal of time to examining object-oriented programming, design, and reuse. In the process, some abstract data structures are developed, as well as others for shapes, telephones, and so on. Unfortunately, I found some errors and inconsistencies in the code between chapters, and this would be disconcerting to the novice. The big example is the development of a shapes system, and a full listing of the result is given. Incidentally, Coplien does not pull his punches on how rich C++ is as a language, and gives the example of

how "*while* (*cp1++ = *cp2++);" might still be unintelligible to someone at the end of a first C course, but is a fundamental building block of a fluent C programmer. Coplien tackles more of the detailed C++ issues than does Cargill, who perhaps has more to give on object-oriented design. Both books are excellent companions for serious programmers. I suggest that both books be in the library and that students be encouraged to buy one based on availability. Practicing programmers can do the same.

As shown in Table 2, Coplien has exercises, but Cargill has few. Coplien's source is also available on ftp and has been tested on a variety of compilers.

Overall Comparison

Now we have to combine all these comparisons and contemplate an overall decision tree. In fact, there are two trees. The first assumes that the book is intended as a university or college course text. Then we differentiate three types of courses, and different books are applicable in each case. For an algorithms and data structures course, choose Budd, then Model, and for advanced programming, choose Wang, then Winder. (Remember that Model is somewhat idiosyncratic and Winder is not for novices.) For a systems course that involves interfacing to hardware, the actual compiler being used is a factor, and one should try to match the book to the software the students will be using: Gorlen for AT&T, Winder for Borland, and Smith for Zortech.

The other need for a C++ book is by a programmer in the field who has been told to learn the language and needs a book on the desk. Although I was not enthusiastic about Lippman, at least it does cover the entire language (which several other books do not), and it is easy to look things up in. So it serves as a good reference manual. For learning the language, however, Wang is still the best. On the other hand, if you already know C, it is likely that you have an investment in some compiler, and so the appropriate book could be chosen to match (see Table 2). Finally, improving one's programming abilities could be done with any of the more advanced books, but I would only recommend those that do so with templates, leading us to Coplien, then Cargill for style and Winder for systems.

Conclusion

There is a place for every one of these books. Each has something to offer, and if one is warned of the shortcomings and does not mind them, then even the books that seemed unattractive at first glance have a place. Looking to the future, the approach that is still missing is that which covers the data structures course and all of C++. I am sure it is possible to do it, and maybe in a couple of years time we shall see such books coming out.

Reviewer: Judith M. Bishop Review #: CR123518 (9507-0447)

Comparative Review

This review compares the following items:

- C++ programming style:
- Developing C++ software (2nd ed.):

- Classic data structures in C++:
- Data abstraction and object-oriented programming in C++:
- C++ primer for C programmers:
- C++ with object-oriented programming:
- Advanced C++:
- Reusability and software construction: C and C++:

Would you recommend this review? yes no Enter

Other reviews under "**C++**": **Date**

Information flow tracking meets just-in-time compilation — Feb 26 2014
Kerschbaumer C., Hennigan E., Larsen P., Brunthaler S., Franz M. ACM Transactions on Architecture and Code Optimization (TACO) 10(4): 1-25, 2013. Type: Article

Challenges and opportunities for learning analytics when formal teaching meets social spaces — Jan 16 2013
Rahman N., Dron J. LAK 2012 (Proceedings of the 2nd International Conference on Learning Analytics and Knowledge, Vancouver, BC, Canada, Apr 29-May 2, 2012) 54-58, 2012. Type: Proceedings

Java generics adoption: how new features are introduced, championed, or ignored — Oct 15 2012
Parnin C., Bird C., Murphy-Hill E. MSR 2011 (Proceeding of the 8th Working Conference on Mining Software Repositories, Waikiki, Honolulu, HI, May 21-22, 2011) 3-12, 2011. Type: Proceedings

more...

E-Mail This Printer-Friendly

REVIEWER'S AREA | MASTHEAD | SUBSCRIBE | PRESS | TIPS | HELP | CONTACT US

Select Language Powered by Google Translate