Scientific Programming 17 (2009) 339–341
DOI 10.3233/SPR-2009-0294
IOS Press

Book Review

Lee S. Brownston [*]
Stinger Ghaffarian Technologies, Inc., USA

Programming: Principles and Practice Using C++, by Bjarne Stroustrup, Pearson Education, Addison–Wesley Professional, 2008, ISBN 978-0321543721.

This is an introduction to computer programming using the C++ language. It's an unusual notion; C++ is commonly thought to be too difficult to teach as a first programming language, that you have to know everything to do anything. Bjarne Stroustrup, the creator of C++ and the major influence on its evolution, disagrees, and he is not about to cede the field to Java, Python and Visual Basic.

The text is organized around several extended example programs that span multiple chapters. The programs grow by successive approximations. Sometimes the examples are purposefully false starts, to show why a naive approach wouldn't work. These extended examples are a "calculator" (algebraic expression evaluator) in Chapters 6 and 7; a 2D graphics package and GUI widget set built on the FLTK toolkit in Chapters 12 through 16; and an STL-like vector class in Chapters 19 and 20. In addition, there are many smaller examples from a variety of domains, including word counting (Chapter 3); a date class (Chapter 9); linked lists (Chapter 17); text editing (Chapter 20); linear algebra (Chapter 24); and encryption (Chapter 25).

The example programs are carefully and imaginatively chosen to introduce programming language elements, computer science concepts and good ("professional") practices. There are few promissory notes in which an example program depends on language features that have not yet been introduced, but there are some. For example, to avoid dealing with exceptions and namespaces, early examples #include a custom-built header file named "std_lib_facilties.h" which hides those details.

Language features are introduced as needed to solve the example programming problems. Sometimes, a minimal amount of information is presented. On page 733, in the discussion of STL algorithm **find_if**(), a function pointer is discreetly presented in a code example, and it is not even called a function pointer, much less is its type mentioned. Then the subject is quickly passed over to a discussion of function objects, though pointers to functions do re-appear in later examples.

Similarly, principles of good programming practice are integrated into each chapter and not isolated in their own sections. The advice is always clear, sound, necessary and undogmatic, but it is not the last word. In fact, Stroustrup's thinking on these matters continues to evolve. For example, on page 597 of the textbook, the recommendation is to pass objects by value if they are tiny; by pointer if the null pointer is a meaningful value; and by reference otherwise. But on page 99 of *The C++ Programming Language: Special Edition*, pointers were preferred over references.

Although the text assumes no prior familiarity with any other programming language, part of Chapter 25 does assume a knowledge of binary and hexadecimal number systems, and some knowledge of linear algebra is needed to understand a matrix example in Chapter 24. The recursive-descent expression parser in the "calculator" example of Chapters 6 and 7 can be pretty rough going for someone encountering these ideas for the first time.

Stroustrup is a precise and careful writer, but the style is conversational rather than formal. On page 802 he acknowledges Brian Kernighan's style as an inspiration, especially "the tutorial sections of his masterpiece, *The C Programming Language*". As with any technical book of such a size (1236 + xxvii pages), there are some errata. I found a few dozen, often having to do with text and examples becoming out of sync. Few are likely to cause problems for the reader, though the characterization of stack behavior as "first in, first out" on page 287 should be corrected by teachers adopting the book.

One point firmly and repeatedly stressed is that programmers should use the standard library containers and algorithms whenever possible, and should not

[*] Address for correspondence: Lee Brownston, Mail Stop 269-3, NASA Ames Research Center, Moffett Field, CA 94035, USA. E-mail: Lee.S.Brownston@nasa.gov.

try to roll their own. This leads to a de-emphasis on data structures and especially algorithms in the text. The implementation of a vector class is presented in detail; linked lists in somewhat less detail; binary trees in even less detail; and stacks are briefly described. Queues are just mentioned and graphs not mentioned at all. The only standard algorithms to be discussed are search: linear, binary and balanced binary trees. Sorting algorithms are not investigated at all: there is just a table of standard library sorting algorithms on page 1117 in an appendix.

The core material in the first 21 chapters is separated from the discretionary material by a charming history of programming languages from 1948 to the near future. Counteracting the abstractness of the technical material, this history, with photographs of the most important innovators, personalizes the enterprise, and places modern programming practice in its intellectual tradition. It is no surprise that there is only grudging mention of Java; in fact, C# is mentioned more frequently.

As with many other computer texts, the discretionary material comes at the end, assumes considerably more background and requires more effort on the part of the student, and is comparatively rushed. The incremental presentation of necessary background diminishes, and the subject matter turns to more advanced and infrequently-encountered topics such as regular expressions, linear algebra, bit manipulations, fragmentation of the free store, real-time embedded programming, and coding standards. The penultimate chapter returns to the important topic of testing, and a final chapter covers the parts of the C language not covered in the preceding chapters because they are not recommended in C++ programs.

There is little treatment of software engineering or the problems specific to large programs, other than the obvious one that the larger the program, the harder it is to write correctly, to test and to debug. There is no discussion of design tools such as UML diagrams. Concepts such as iterative development and agile programming are absent. The examples are all small programs and the point of view is usually that of someone responsible for a small unit of functionality, a few classes at most. Although there are abstract base classes, the concept of a class used as an interface is not presented.

Similarly, many established concepts and terms are avoided. In passing, mention is made of writing a class the sole purpose of which is to supply a new interface, but the terms "adaptor", "facade" and "proxy" are not used, nor is there any other allusion to design patterns. In discussing the testing of GUIs on pages 969–973, programmers are recommended to separate I/O, whether GUI or text I/O, from the main program, but there is no allusion to the concept of Model–View–Controller. These absences are emblematic of the insularity of the C++ inner circle. Not only does the C++ community insist on its own vocabulary ("base class" instead of "superclass"; "member function" instead of "method"), but the most authoritative writings seldom make reference to ideas which originated outside of the C++ community itself or the direct ancestors of C++, such as Simula and, of course, C. Students will encounter these ideas if they continue their computer science studies, but a teacher would do them a service by presenting these ideas in class.

This text shares many of the features common among computer texts. The formatting involves gobs of white space, since turning pages rapidly gives the reader the feeling of making rapid progress: big margins, especially around code examples, bulleted lists, tables and diagrams; chapters and parts beginning with sparsely-populated pages. Chapters and appendices have numbered sections and sometimes subsections, and the table of contents enumerates them all, taking 18 pages to do so. Small colored disks in the margin highlight important points, and the exposition is sometimes interrupted by "Try this" suggestions. Each chapter ends with "drills" (exercises which don't involve much original thought); review questions; a list of terms introduced in the chapter; an extensive set of exercises; and a "postscript" that comments on the material just presented and often looks forward to the following chapter. Color is used extensively in diagrams, tables, icons, chapter beginnings and endings and color photographs of programming language innovators.

The chapter on testing emphasizes the choice of tests to run and presents a roll-your-own approach to test suite construction. It's odd that unit test frameworks such as the open-source CppUnit port of JUnit are not even mentioned, especially considering the strong deprecation of duplicating standard library data structures and algorithms. In fact, FLTK and Visual Studio are the only products considered. A brief survey of the kinds of tools a programmer might want to use would have been welcome.

Although Stroustrup warns against putting too much emphasis on formatting and other stylistic matters, the code examples implicitly present a consistent coding style: under_scores rather than CamelCase; extremely short variable and formal parameter names; no lexical distinction between class and struct data member names on the one hand and variable and formal parameter names on the other; absence of curly braces around single-statement blocks in **if**, **for** and **while** statements; the asterisk for a pointer is attached to the type name rather than to the variable or formal parameter; header comments appear between the signature and the body; **const** is used when needed but not everywhere it is possible. All of these are defensible choices, and may even be the best choices, but alternatives are rarely discussed (e.g., the author's personal distaste for camel case on page 936).

Stroustrup is justly proud of the success of C++ and of his role in its creation and development. This tends to make him soft-pedal the tradeoffs involved in choosing a language. While acknowledging that no language does everything equally well, he is far from specific about the shortcomings of C++. To be sure, a textbook author doesn't want to discourage students about the value of what they are studying, but a bit more candor about the tradeoffs would have been welcome. For example, among all the numerous application domains mentioned in the text, web application development is missing. This is an area in which PHP, Java, Python, Visual Basic and even Perl are more popular than C++. On page 773 is a list of desirable properties of programming languages. Missing from this list are rapid development, short development cycles, modification without re-compilation and protection against memory leaks. So we have a chicken-and-egg problem: does he value C++ because it has most of these properties, or does he value these properties because C++ possesses so many of them? I suspect some of both.

Experienced programmers, even experienced C++ programmers, can profit from reading *Programming: Principles and Practice Using C++*. They can learn about some of the less-familiar language features, some perspicuous uses of more-familiar language features and good programming practices, as well as simply being exposed to several carefully-written programs designed by an authority in the field. *The C++ Programming Language* will give advanced readers deeper and more comprehensive coverage, but there is still value for them in this book as a refresher.

While reading this book, I couldn't help thinking of the Caltech students who studied introductory physics from Richard Feynman's famed lectures. No, the demands on the student are not nearly as great, nor is the interest value for the experienced. This textbook was developed over several years at Texas A & M University, and this is a credit to the level of students for whom this was their introduction to programming. It is unlikely that this text will be adopted in community colleges and university extension courses.