Preface

All problems in computer science
can be solved by another level of indirection,
except for the problem of too many layers of indirection.
– David J. Wheeler

C++ feels like a new language. That is, I can express my ideas more clearly, more simply, and more directly in C++11 than I could in C++98. Furthermore, the resulting programs are better checked by the compiler and run faster.

In this book, I aim for *completeness*. I describe every language feature and standard-library component that a professional programmer is likely to need. For each, I provide:

- *Rationale*: What kinds of problems is it designed to help solve? What principles underlie the design? What are the fundamental limitations?
- *Specification*: What is its definition? The level of detail is chosen for the expert programmer; the aspiring language lawyer can follow the many references to the ISO standard.
- *Examples*: How can it be used well by itself and in combination with other features? What are the key techniques and idioms? What are the implications for maintainability and performance?

The use of C++ has changed dramatically over the years and so has the language itself. From the point of view of a programmer, most of the changes have been improvements. The current ISO standard C++ (ISO/IEC 14882-2011, usually called C++11) is simply a far better tool for writing quality software than were previous versions. How is it a better tool? What kinds of programming styles and techniques does modern C++ support? What language and standard-library features support those techniques? What are the basic building blocks of elegant, correct, maintainable, and efficient C++ code? Those are the key questions answered by this book. Many answers are not the same as you would find with 1985, 1995, or 2005 vintage C++: progress happens.

C++ is a general-purpose programming language emphasizing the design and use of type-rich, lightweight abstractions. It is particularly suited for resource-constrained applications, such as those found in software infrastructures. C++ rewards the programmer who takes the time to master

techniques for writing quality code. C++ is a language for someone who takes the task of programming seriously. Our civilization depends critically on software; it had better be quality software.

There are billions of lines of C++ deployed. This puts a premium on stability, so 1985 and 1995 C++ code still works and will continue to work for decades. However, for all applications, you can do better with modern C++; if you stick to older styles, you will be writing lower-quality and worse-performing code. The emphasis on stability also implies that standards-conforming code you write today will still work a couple of decades from now. All code in this book conforms to the 2011 ISO C++ standard.

This book is aimed at three audiences:

- C++ programmers who want to know what the latest ISO C++ standard has to offer,
- C programmers who wonder what C++ provides beyond C, and
- People with a background in application languages, such as Java, C#, Python, and Ruby, looking for something ''closer to the machine'' – something more flexible, something offering better compile-time checking, or something offering better performance.

Naturally, these three groups are not disjoint – a professional software developer masters more than just one programming language.

This book assumes that its readers are programmers. If you ask, ''What's a for-loop?'' or ''What's a compiler?'' then this book is not (yet) for you; instead, I recommend my *Programming: Principles and Practice Using C++* to get started with programming and C++. Furthermore, I assume that readers have some maturity as software developers. If you ask ''Why bother testing?'' or say, ''All languages are basically the same; just show me the syntax'' or are confident that there is a single language that is ideal for every task, this is not the book for you.

What features does C++11 offer over and above C++98? A machine model suitable for modern computers with lots of concurrency. Language and standard-library facilities for doing systems-level concurrent programming (e.g., using multicores). Regular expression handling, resource management pointers, random numbers, improved containers (including, hash tables), and more. General and uniform initialization, a simpler **for**-statement, move semantics, basic Unicode support, lambdas, general constant expressions, control over class defaults, variadic templates, user-defined literals, and more. Please remember that those libraries and language features exist to support programming techniques for developing quality software. They are meant to be used in combination – as bricks in a building set – rather than to be used individually in relative isolation to solve a specific problem. A computer is a universal machine, and C++ serves it in that capacity. In particular, C++'s design aims to be sufficiently flexible and general to cope with future problems undreamed of by its designers.

Acknowledgments

In addition to the people mentioned in the acknowledgment sections of the previous editions, I would like to thank Pete Becker, Hans-J. Boehm, Marshall Clow, Jonathan Coe, Lawrence Crowl, Walter Daugherty, J. Daniel Garcia, Robert Harle, Greg Hickman, Howard Hinnant, Brian Kernighan, Daniel Krügler, Nevin Liber, Michel Michaud, Gary Powell, Jan Christiaan van Winkel, and Leor Zolman. Without their help this book would have been much poorer.

Thanks to Howard Hinnant for answering many questions about the standard library.

Andrew Sutton is the author of the Origin library, which was the testbed for much of the discussion of emulating concepts in the template chapters, and of the matrix library that is the topic of Chapter 29. The Origin library is open source and can be found by searching the Web for ‘‘Origin’’ and ‘‘Andrew Sutton.’’

Thanks to my graduate design class for finding more problems with the ‘‘tour chapters’’ than anyone else.

Had I been able to follow every piece of advice of my reviewers, the book would undoubtedly have been much improved, but it would also have been hundreds of pages longer. Every expert reviewer suggested adding technical details, advanced examples, and many useful development conventions; every novice reviewer (or educator) suggested adding examples; and most reviewers observed (correctly) that the book may be too long.

Thanks to Princeton University’s Computer Science Department, and especially Prof. Brian Kernighan, for hosting me for part of the sabbatical that gave me time to write this book.

Thanks to Cambridge University’s Computer Lab, and especially Prof. Andy Hopper, for hosting me for part of the sabbatical that gave me time to write this book.

Thanks to my editor, Peter Gordon, and his production team at Addison-Wesley for their help and patience.

College Station, Texas *Bjarne Stroustrup*