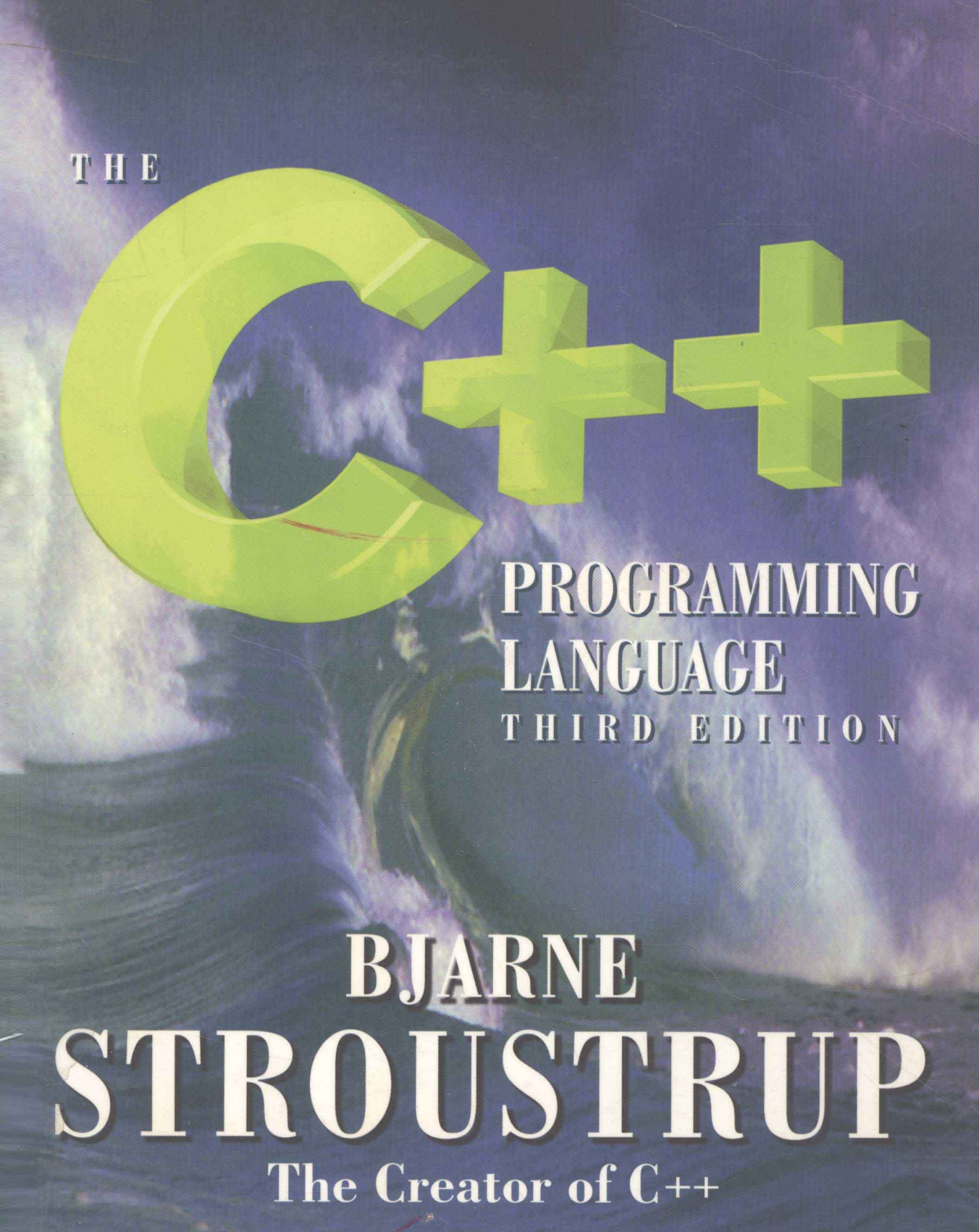
THE
C++
PROGRAMMING
LANGUAGE
THIRD EDITION
BJARNE
STROUSTRUP
The Creator of C++

The C++ Programming Language

Third Edition

Bjarne Stroustrup

AT&T Labs
Murray Hill, New Jersey

Addison-Wesley
An Imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts · Harlow, England · Menlo Park, California
Berkeley, California · Don Mills, Ontario · Sydney
Bonn · Amsterdam · Tokyo · Mexico City

The publisher offers discounts on this book when ordered in quantity for special sales. For more information please contact:

Corporate & Professional Publishing Group
Addison-Wesley Publishing Company
One Jacob Way
Reading, Massachusetts 01867

Library of Congress Cataloging-in-Publication Data

Stroustrup, Bjarne
The C++ Programming Language / Bjarne Stroustrup. — 3rd. ed.
p. cm.
Includes index.
ISBN 0-201-88954-4
1. C++ (Computer Programming Language) I. Title
QA76.73.C153S77 1997 97-20239
005.13'3—dc21 CIP

This book was typeset in Times and Courier by the author.

Text printed on recycled and acid-free paper.
ISBN 0-201-88954-4
9 1011121314 CRS 02 01 00 99
9th Printing January 1999

Preface

Programming is understanding.
– Kristen Nygaard

I find using C++ more enjoyable than ever. C++'s support for design and programming has improved dramatically over the years, and lots of new helpful techniques have been developed for its use. However, C++ is not *just* fun. Ordinary practical programmers have achieved significant improvements in productivity, maintainability, flexibility, and quality in projects of just about any kind and scale. By now, C++ has fulfilled most of the hopes I originally had for it, and also succeeded at tasks I hadn't even dreamt of.

This book introduces standard C++† and the key programming and design techniques supported by C++. Standard C++ is a far more powerful and polished language than the version of C++ introduced by the first edition of this book. New language features such as namespaces, exceptions, templates, and run-time type identification allow many techniques to be applied more directly than was possible before, and the standard library allows the programmer to start from a much higher level than the bare language.

About a third of the information in the second edition of this book came from the first. This third edition is the result of a rewrite of even larger magnitude. It offers something to even the most experienced C++ programmer; at the same time, this book is easier for the novice to approach than its predecessors were. The explosion of C++ use and the massive amount of experience accumulated as a result makes this possible.

The definition of an extensive standard library makes a difference to the way C++ concepts can be presented. As before, this book presents C++ independently of any particular implementation, and as before, the tutorial chapters present language constructs and concepts in a "bottom up" order so that a construct is used only after it has been defined. However, it is much easier to use a well-designed library than it is to understand the details of its implementation. Therefore, the standard library can be used to provide realistic and interesting examples well before a reader can be assumed to understand its inner workings. The standard library itself is also a fertile source of programming examples and design techniques.

---

† ISO/IEC 14882, Standard for the C++ Programming Language.

This book presents every major C++ language feature and the standard library. It is organized around language and library facilities. However, features are presented in the context of their use. That is, the focus is on the language as the tool for design and programming rather than on the language in itself. This book demonstrates key techniques that make C++ effective and teaches the fundamental concepts necessary for mastery. Except where illustrating technicalities, examples are taken from the domain of systems software. A companion, *The Annotated C++ Language Standard*, presents the complete language definition together with annotations to make it more comprehensible.

The primary aim of this book is to help the reader understand how the facilities offered by C++ support key programming techniques. The aim is to take the reader far beyond the point where he or she gets code running primarily by copying examples and emulating programming styles from other languages. Only a good understanding of the ideas behind the language facilities leads to mastery. Supplemented by implementation documentation, the information provided is sufficient for completing significant real-world projects. The hope is that this book will help the reader gain new insights and become a better programmer and designer.

Acknowledgments

In addition to the people mentioned in the acknowledgement sections of the first and second editions, I would like to thank Matt Austern, Hans Boehm, Don Caldwell, Lawrence Crowl, Alan Feuer, Andrew Forrest, David Gay, Tim Griffin, Peter Juhl, Brian Kernighan, Andrew Koenig, Mike Mowbray, Rob Murray, Lee Nackman, Joseph Newcomer, Alex Stepanov, David Vandevoorde, Peter Weinberger, and Chris Van Wyk for commenting on draft chapters of this third edition. Without their help and suggestions, this book would have been harder to understand, contained more errors, been slightly less complete, and probably been a little bit shorter.

I would also like to thank the volunteers on the C++ standards committees who did an immense amount of constructive work to make C++ what it is today. It is slightly unfair to single out individuals, but it would be even more unfair not to mention anyone, so I'd like to especially mention Mike Ball, Dag Brück, Sean Corfield, Ted Goldstein, Kim Knuttila, Andrew Koenig, Josée Lajoie, Dmitry Lenkov, Nathan Myers, Martin O'Riordan, Tom Plum, Jonathan Shopiro, John Spicer, Jerry Schwarz, Alex Stepanov, and Mike Vilot, as people who each directly cooperated with me over some part of C++ and its standard library.

Murray Hill, New Jersey *Bjarne Stroustrup*

Preface to the Second Edition

The road goes ever on and on.
– Bilbo Baggins

As promised in the first edition of this book, C++ has been evolving to meet the needs of its users. This evolution has been guided by the experience of users of widely varying backgrounds working in a great range of application areas. The C++ user-community has grown a hundredfold during the six years since the first edition of this book; many lessons have been learned, and many techniques have been discovered and/or validated by experience. Some of these experiences are reflected here.

The primary aim of the language extensions made in the last six years has been to enhance C++ as a language for data abstraction and object-oriented programming in general and to enhance it as a tool for writing high-quality libraries of user-defined types in particular. A ''high-quality library,'' is a library that provides a concept to a user in the form of one or more classes that are convenient, safe, and efficient to use. In this context, *safe* means that a class provides a specific type-safe interface between the users of the library and its providers; *efficient* means that use of the class does not impose significant overheads in run-time or space on the user compared with hand-written C code.

This book presents the complete C++ language. Chapters 1 through 10 give a tutorial introduction; Chapters 11 through 13 provide a discussion of design and software development issues; and, finally, the complete C++ reference manual is included. Naturally, the features added and resolutions made since the original edition are integral parts of the presentation. They include refined overloading resolution, memory management facilities, and access control mechanisms, type-safe linkage, ***const*** and ***static*** member functions, abstract classes, multiple inheritance, templates, and exception handling.

C++ is a general-purpose programming language; its core application domain is systems programming in the broadest sense. In addition, C++ is successfully used in many application areas that are not covered by this label. Implementations of C++ exist from some of the most modest microcomputers to the largest supercomputers and for almost all operating systems. Consequently, this book describes the C++ language itself without trying to explain a particular implementation, programming environment, or library.

This book presents many examples of classes that, though useful, should be classified as ''toys.'' This style of exposition allows general principles and useful techniques to stand out more

clearly than they would in a fully elaborated program, where they would be buried in details. Most of the useful classes presented here, such as linked lists, arrays, character strings, matrices, graphics classes, associative arrays, etc., are available in ''bulletproof'' and/or ''goldplated'' versions from a wide variety of commercial and non-commercial sources. Many of these ''industrial strength'' classes and libraries are actually direct and indirect descendants of the toy versions found here.

This edition provides a greater emphasis on tutorial aspects than did the first edition of this book. However, the presentation is still aimed squarely at experienced programmers and endeavors not to insult their intelligence or experience. The discussion of design issues has been greatly expanded to reflect the demand for information beyond the description of language features and their immediate use. Technical detail and precision have also been increased. The reference manual, in particular, represents many years of work in this direction. The intent has been to provide a book with a depth sufficient to make more than one reading rewarding to most programmers. In other words, this book presents the C++ language, its fundamental principles, and the key techniques needed to apply it. Enjoy!

Acknowledgments

In addition to the people mentioned in the acknowledgements section in the preface to the first edition, I would like to thank Al Aho, Steve Buroff, Jim Coplien, Ted Goldstein, Tony Hansen, Lorraine Juhl, Peter Juhl, Brian Kernighan, Andrew Koenig, Bill Leggett, Warren Montgomery, Mike Mowbray, Rob Murray, Jonathan Shopiro, Mike Vilot, and Peter Weinberger for commenting on draft chapters of this second edition. Many people influenced the development of C++ from 1985 to 1991. I can mention only a few: Andrew Koenig, Brian Kernighan, Doug McIlroy, and Jonathan Shopiro. Also thanks to the many participants of the ''external reviews'' of the reference manual drafts and to the people who suffered through the first year of X3J16.

Murray Hill, New Jersey *Bjarne Stroustrup*

Preface to the First Edition

Language shapes the way we think,
and determines what we can think about.
– B.L.Whorf

C++ is a general purpose programming language designed to make programming more enjoyable for the serious programmer. Except for minor details, C++ is a superset of the C programming language. In addition to the facilities provided by C, C++ provides flexible and efficient facilities for defining new types. A programmer can partition an application into manageable pieces by defining new types that closely match the concepts of the application. This technique for program construction is often called *data abstraction.* Objects of some user-defined types contain type information. Such objects can be used conveniently and safely in contexts in which their type cannot be determined at compile time. Programs using objects of such types are often called *object based.* When used well, these techniques result in shorter, easier to understand, and easier to maintain programs.

The key concept in C++ is *class.* A class is a user-defined type. Classes provide data hiding, guaranteed initialization of data, implicit type conversion for user-defined types, dynamic typing, user-controlled memory management, and mechanisms for overloading operators. C++ provides much better facilities for type checking and for expressing modularity than C does. It also contains improvements that are not directly related to classes, including symbolic constants, inline substitution of functions, default function arguments, overloaded function names, free store management operators, and a reference type. C++ retains C's ability to deal efficiently with the fundamental objects of the hardware (bits, bytes, words, addresses, etc.). This allows the user-defined types to be implemented with a pleasing degree of efficiency.

C++ and its standard libraries are designed for portability. The current implementation will run on most systems that support C. C libraries can be used from a C++ program, and most tools that support programming in C can be used with C++.

This book is primarily intended to help serious programmers learn the language and use it for nontrivial projects. It provides a complete description of C++, many complete examples, and many more program fragments.

Acknowledgments

C++ could never have matured without the constant use, suggestions, and constructive criticism of many friends and colleagues. In particular, Tom Cargill, Jim Coplien, Stu Feldman, Sandy Fraser, Steve Johnson, Brian Kernighan, Bart Locanthi, Doug McIlroy, Dennis Ritchie, Larry Rosler, Jerry Schwarz, and Jon Shopiro provided important ideas for development of the language. Dave Presotto wrote the current implementation of the stream I/O library.

In addition, hundreds of people contributed to the development of C++ and its compiler by sending me suggestions for improvements, descriptions of problems they had encountered, and compiler errors. I can mention only a few: Gary Bishop, Andrew Hume, Tom Karzes, Victor Milenkovic, Rob Murray, Leonie Rose, Brian Schmult, and Gary Walker.

Many people have also helped with the production of this book, in particular, Jon Bentley, Laura Eaves, Brian Kernighan, Ted Kowalski, Steve Mahaney, Jon Shopiro, and the participants in the C++ course held at Bell Labs, Columbus, Ohio, June 26-27, 1985.

Murray Hill, New Jersey *Bjarne Stroustrup*

Contents

Introduction

This introduction gives an overview of the major concepts and features of the C++ programming language and its standard library. It also provides an overview of this book and explains the approach taken to the description of the language facilities and their use. In addition, the introductory chapters present some background information about C++, the design of C++, and the use of C++.

Chapters

Introduction

''... and you, Marcus, you have given me many things; now I shall give you this good advice. Be many people. Give up the game of being always Marcus Cocoza. You have worried too much about Marcus Cocoza, so that you have been really his slave and prisoner. You have not done anything without first considering how it would affect Marcus Cocoza's happiness and prestige. You were always much afraid that Marcus might do a stupid thing, or be bored. What would it really have mattered? All over the world people are doing stupid things ... I should like you to be easy, your little heart to be light again. You must from now, be more than one, many people, as many as you can think of ...''

– Karen Blixen
(''The Dreamers'' from ''Seven Gothic Tales''
written under the pseudonym Isak Dinesen,
Random House, Inc.

1

Notes to the Reader

"The time has come," the Walrus said,
"to talk of many things."
– L.Carroll

Structure of this book — how to learn C++ — the design of C++ — efficiency and structure — philosophical note — historical note — what C++ is used for — C and C++ — suggestions for C programmers — suggestions for C++ programmers — thoughts about programming in C++ — advice — references.

1.1 The Structure of This Book

This book consists of six parts:

Introduction: Chapters 1 through 3 give an overview of the C++ language, the key programming styles it supports, and the C++ standard library.

Part I: Chapters 4 through 9 provide a tutorial introduction to C++'s built-in types and the basic facilities for constructing programs out of them.

Part II: Chapters 10 through 15 are a tutorial introduction to object-oriented and generic programming using C++.

Part III: Chapters 16 through 22 present the C++ standard library.

Part IV: Chapters 23 through 25 discuss design and software development issues.

Appendices: Appendices A through C provide language-technical details.

Chapter 1 provides an overview of this book, some hints about how to use it, and some background information about C++ and its use. You are encouraged to skim through it, read what appears interesting, and return to it after reading other parts of the book.

Chapters 2 and 3 provide an overview of the major concepts and features of the C++ programming language and its standard library. Their purpose is to motivate you to spend time on fundamental concepts and basic language features by showing what can be expressed using the complete

C++ language. If nothing else, these chapters should convince you that C++ isn't (just) C and that C++ has come a long way since the first and second editions of this book. Chapter 2 gives a high-level acquaintance with C++. The discussion focuses on the language features supporting data abstraction, object-oriented programming, and generic programming. Chapter 3 introduces the basic principles and major facilities of the standard library. This allows me to use standard library facilities in the following chapters. It also allows you to use library facilities in exercises rather than relying directly on lower-level, built-in features.

The introductory chapters provide an example of a general technique that is applied throughout this book: to enable a more direct and realistic discussion of some technique or feature, I occasionally present a concept briefly at first and then discuss it in depth later. This approach allows me to present concrete examples before a more general treatment of a topic. Thus, the organization of this book reflects the observation that we usually learn best by progressing from the concrete to the abstract – even where the abstract seems simple and obvious in retrospect.

Part I describes the subset of C++ that supports the styles of programming traditionally done in C or Pascal. It covers fundamental types, expressions, and control structures for C++ programs. Modularity – as supported by namespaces, source files, and exception handling – is also discussed. I assume that you are familiar with the fundamental programming concepts used in Part I. For example, I explain C++'s facilities for expressing recursion and iteration, but I do not spend much time explaining how these concepts are useful.

Part II describes C++'s facilities for defining and using new types. Concrete and abstract classes (interfaces) are presented here (Chapter 10, Chapter 12), together with operator overloading (Chapter 11), polymorphism, and the use of class hierarchies (Chapter 12, Chapter 15). Chapter 13 presents templates, that is, C++'s facilities for defining families of types and functions. It demonstrates the basic techniques used to provide containers, such as lists, and to support generic programming. Chapter 14 presents exception handling, discusses techniques for error handling, and presents strategies for fault tolerance. I assume that you either aren't well acquainted with object-oriented programming and generic programming or could benefit from an explanation of how the main abstraction techniques are supported by C++. Thus, I don't just present the language features supporting the abstraction techniques; I also explain the techniques themselves. Part IV goes further in this direction.

Part III presents the C++ standard library. The aim is to provide an understanding of how to use the library, to demonstrate general design and programming techniques, and to show how to extend the library. The library provides containers (such as *list*, *vector*, and *map*; Chapter 16, Chapter 17), standard algorithms (such as *sort*, *find*, and *merge*; Chapter 18, Chapter 19), strings (Chapter 20), Input/Output (Chapter 21), and support for numerical computation (Chapter 22).

Part IV discusses issues that arise when C++ is used in the design and implementation of large software systems. Chapter 23 concentrates on design and management issues. Chapter 24 discusses the relation between the C++ programming language and design issues. Chapter 25 presents some ways of using classes in design.

Appendix A is C++'s grammar, with a few annotations. Appendix B discusses the relation between C and C++ and between Standard C++ (also called ISO C++ and ANSI C++) and the versions of C++ that preceded it. Appendix C presents some language-technical examples.

1.1.1 Examples and References

This book emphasizes program organization rather than the writing of algorithms. Consequently, I avoid clever or harder-to-understand algorithms. A trivial algorithm is typically better suited to illustrate an aspect of the language definition or a point about program structure. For example, I use a Shell sort where, in real code, a quicksort would be better. Often, reimplementation with a more suitable algorithm is an exercise. In real code, a call of a library function is typically more appropriate than the code used here for illustration of language features.

Textbook examples necessarily give a warped view of software development. By clarifying and simplifying the examples, the complexities that arise from scale disappear. I see no substitute for writing realistically-sized programs for getting an impression of what programming and a programming language are really like. This book concentrates on the language features, the basic techniques from which every program is composed, and the rules for composition.

The selection of examples reflects my background in compilers, foundation libraries, and simulations. Examples are simplified versions of what is found in real code. The simplification is necessary to keep programming language and design points from getting lost in details. There are no "cute" examples without counterparts in real code. Wherever possible, I relegated to Appendix C language-technical examples of the sort that use variables named *x* and *y*, types called *A* and *B*, and functions called *f*() and *g*().

In code examples, a proportional-width font is used for identifiers. For example:

```
#include<iostream>

int main()
{
    std::cout << "Hello, new world!\n";
}
```

At first glance, this presentation style will seem "unnatural" to programmers accustomed to seeing code in constant-width fonts. However, proportional-width fonts are generally regarded as better than constant-width fonts for presentation of text. Using a proportional-width font also allows me to present code with fewer illogical line breaks. Furthermore, my experiments show that most people find the new style more readable after a short while.

Where possible, the C++ language and library features are presented in the context of their use rather than in the dry manner of a manual. The language features presented and the detail in which they are described reflect my view of what is needed for effective use of C++. A companion, *The Annotated C++ Language Standard*, authored by Andrew Koenig and myself, is the complete definition of the language together with comments aimed at making it more accessible. Logically, there ought to be another companion, *The Annotated C++ Standard Library*. However, since both time and my capacity for writing are limited, I cannot promise to produce that.

References to parts of this book are of the form §2.3.4 (Chapter 2, section 3, subsection 4), §B.5.6 (Appendix B, subsection 5.6), and §6.6[10] (Chapter 6, exercise 10). Italics are used sparingly for emphasis (e.g., "a string literal is *not* acceptable"), for first occurrences of important concepts (e.g., *polymorphism*), for nonterminals of the C++ grammar (e.g., *for-statement*), and for comments in code examples. Semi-bold italics are used to refer to identifiers, keywords, and numeric values from code examples (e.g., ***class***, ***counter***, and ***1712***).

1.1.2 Exercises

Exercises are found at the ends of chapters. The exercises are mainly of the write-a-program variety. Always write enough code for a solution to be compiled and run with at least a few test cases. The exercises vary considerably in difficulty, so they are marked with an estimate of their difficulty. The scale is exponential so that if a (*1) exercise takes you ten minutes, a (*2) might take an hour, and a (*3) might take a day. The time needed to write and test a program depends more on your experience than on the exercise itself. A (*1) exercise might take a day if you first have to get acquainted with a new computer system in order to run it. On the other hand, a (*5) exercise might be done in an hour by someone who happens to have the right collection of programs handy.

Any book on programming in C can be used as a source of extra exercises for Part I. Any book on data structures and algorithms can be used as a source of exercises for Parts II and III.

1.1.3 Implementation Note

The language used in this book is ''pure C++'' as defined in the C++ standard [C++,1998]. Therefore, the examples ought to run on every C++ implementation. The major program fragments in this book were tried using several C++ implementations. Examples using features only recently adopted into C++ didn't compile on every implementation. However, I see no point in mentioning which implementations failed to compile which examples. Such information would soon be out of date because implementers are working hard to ensure that their implementations correctly accept every C++ feature. See Appendix B for suggestions on how to cope with older C++ compilers and with code written for C compilers.

1.2 Learning C++

The most important thing to do when learning C++ is to focus on concepts and not get lost in language-technical details. The purpose of learning a programming language is to become a better programmer; that is, to become more effective at designing and implementing new systems and at maintaining old ones. For this, an appreciation of programming and design techniques is far more important than an understanding of details; that understanding comes with time and practice.

C++ supports a variety of programming styles. All are based on strong static type checking, and most aim at achieving a high level of abstraction and a direct representation of the programmer's ideas. Each style can achieve its aims effectively while maintaining run-time and space efficiency. A programmer coming from a different language (say C, Fortran, Smalltalk, Lisp, ML, Ada, Eiffel, Pascal, or Modula-2) should realize that to gain the benefits of C++, they must spend time learning and internalizing programming styles and techniques suitable to C++. The same applies to programmers used to an earlier and less expressive version of C++.

Thoughtlessly applying techniques effective in one language to another typically leads to awkward, poorly performing, and hard-to-maintain code. Such code is also most frustrating to write because every line of code and every compiler error message reminds the programmer that the language used differs from ''the old language.'' You can write in the style of Fortran, C, Smalltalk, etc., in any language, but doing so is neither pleasant nor economical in a language with a different philosophy. Every language can be a fertile source of ideas of how to write C++ programs.