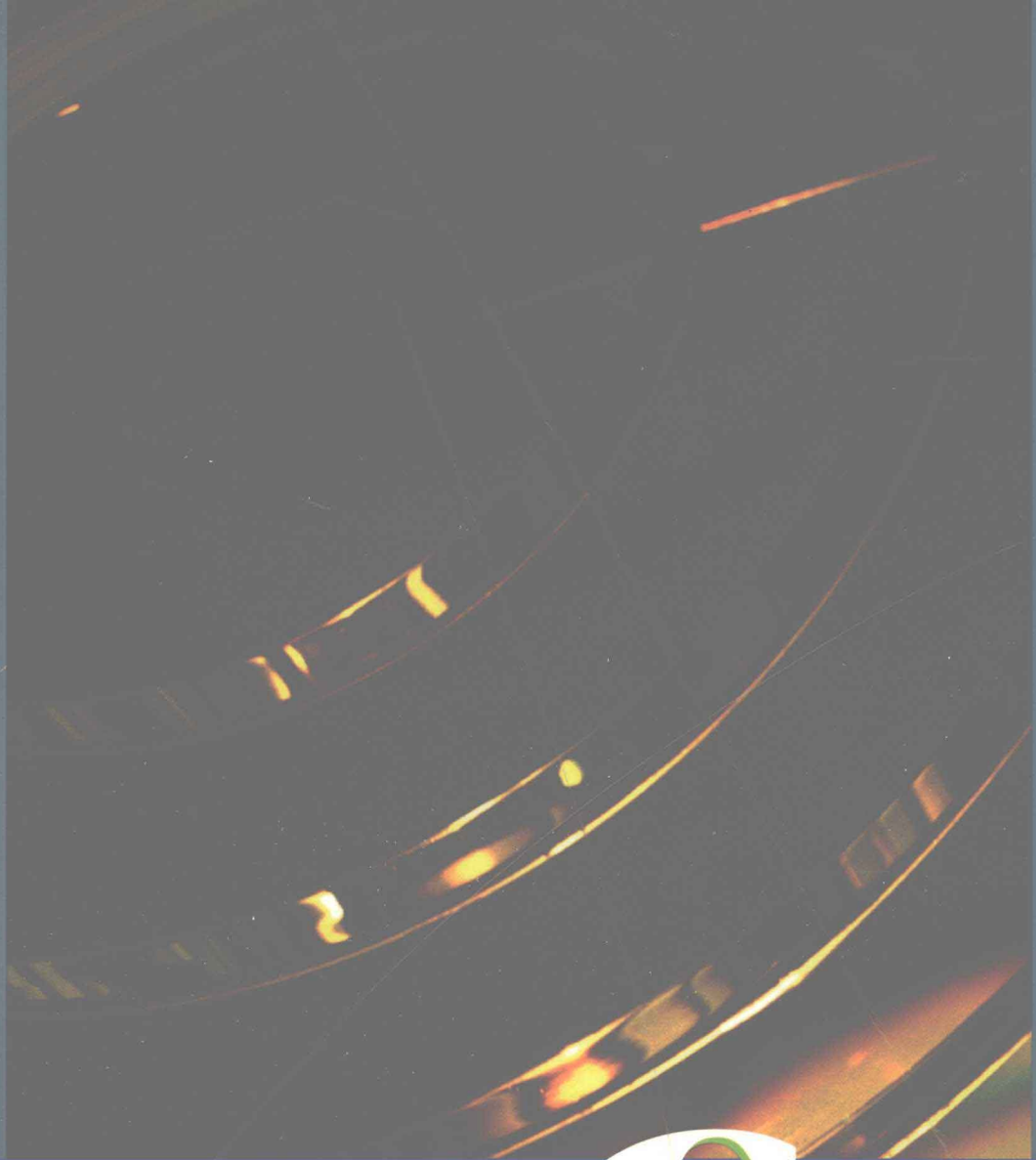

Using C++

An Introduction to Programming

Julien Hennefeld • Charles Burchard

Using C++

An Introduction to Programming

Julien Hennefeld
Brooklyn College of the City University of New York

Charles Burchard
Penn State Erie—The Behrend College

PWS Publishing Company

I(T)P **An International Thomson Publishing Company**

Boston • Albany • Bonn • Cincinnati • London • Melbourne • Mexico City
New York • Paris • San Francisco • Singapore • Tokyo • Toronto • Washington

PWS Publishing Company
20 Park Plaza, Boston, MA 02116–4324

Copyright © 1998 by PWS Publishing Company,
a division of International Thomson Publishing, Inc.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted by any means—electronic, mechanical, photocopying, recording, or otherwise—without prior written permission of PWS Publishing Company.

I(T)P ®
International Thomson Publishing
The trademark ITP is used under license.

Printed in the United States of America.
00 01 — 10 9 8 7 6 5 4

This book is printed on recycled, acid-free paper.

Sponsoring Editor: *David Dietz*
Production Editor: *Andrea Goldman*
Manufacturing Coordinator: *Andrew Christensen*
Marketing Manager: *Nathan Wilbur*
Editorial Assistant: *Kathryn Schooling*
Copyeditors: *Andrea Goldman, Lorretta Palagi*
Composition/Art: *The PRD Group*
Cover Design: *Peter Blaiwas*
Interior Design: *Sandra Rigney*
Text Printer and Binder: *R.R. Donnelley & Sons–Crawfordsville*
Cover Printer: *Phoenix Color Corp.*

Library of Congress Cataloging-in-Publication-Data

Hennefeld, Julien O.
Using C++: an introduction to programming / Julien Hennefeld, Charles Burchard.
p. cm.
Includes index.
ISBN 0–534–95591–6
1. C++ (Computer program language) I. Burchard, Charles. II. Title.
QA76.73.C153H457 1998
005.13'3–dc21 97–31817
CIP

For more information, contact:
PWS Publishing Company
20 Park Plaza
Boston, MA 02116

International Thomson Publishing Europe
Berkshire House
168-173 High Holborn
London WC1V 7AA
England

Thomas Nelson Australia
102 Dodds Street
South Melbourne, 3205
Victoria, Australia

Nelson Canada
1120 Birchmont Road
Scarborough, Ontario
Canada M1K 5G4

International Thomson Editores
Campos Eliseos 385, Piso 7
Col. Polanco
11560 Mexico D.F., Mexico

International Thomson Publishing GmbH
Königswinterer Strasse 418
53227 Bonn, Germany

International Thomson Publishing Asia
221 Henderson Road
#05-10 Henderson Building
Singapore 0315

International Thomson Publishing Japan
Hirakawacho Kyowa Building, 31
2-2-1 Hirakawacho
Chiyoda-ku, Tokyo 102
Japan

Using C++

I would like to thank my wife, Marianne, my children, Maggie and Dan, and my canine associate, Zoe, for bearing with me and sustaining me during the difficult stretches of work on the book.

Julien Hennefeld

I would like to thank my wife, Marge, for "running the ship" while I was off on this venture, and my children, Sarah and Emmy, for providing some laughs when I was taking it all too seriously.

Chuck Burchard

此为试读,需要完整PDF请访问: www.ertongbook.com

Preface

Intended Audience

This book can be used in a number of different types of C++ programming courses. Its clarity, readability, and wealth of illustrative examples make it especially suitable for use in a CS1 course for beginning programmers. Concisely written, the book ultimately is comprehensive and includes more material than can be covered in a single semester by beginners. Thus, it may be used in a single-semester, faster-paced course for computer science majors or students with prior programming experience, or it may be used in a two-semester sequence for beginners and nonmajors. Since it covers almost all of the material recommended for the Advanced Placement A and B exams, it is especially suitable for secondary-level courses. Moreover, it can be used both in courses that limit coverage to procedural C++ and in courses that will eventually cover a significant amount of the object-oriented features of the language.

Content Approach

Basics

At the beginning, we teach C++ as a "better C," because it is an easier and more convenient language in which to write basic programs. We strongly emphasize control structures and writing and designing with functions, and encourage students to build onto a library of useful functions that they can reuse. We discuss problem solving and program design using a four-step method, and make frequent use of pseudocode and top-down development. Because extended, detailed discussions of program development can work well in the classroom but become tedious

and visually overwhelming on the printed page, we keep discussions of program development focused and streamlined.

Should Objects Be Taught Early?

Currently, the question of how early to introduce object orientation in the first course is hotly debated. We believe that it is important for students to become proficient in applying and implementing C++ classes by the end of a two-semester sequence, but our experience has been that discussing the syntax for declaring and implementing C++ classes too early confuses students more than it enlightens them. After all, C++ is a hybrid language, and using its complicated object-oriented overlay effectively presupposes quite a bit of mastery of its procedural features. Furthermore, object orientation's extensive overhead is geared toward safely handling the complexity of large programming projects. For many short and medium-size programming problems, object orientation is not appropriate and, when applied to such problems, does not give a convincing introduction to the power and usefulness of object-oriented programming. Further, we are convinced that object-oriented design is a difficult topic that cannot be taught effectively until students have had ample practice in using and implementing objects.

Early Approach to Objects

We do believe, however, that it is possible to give students a *meaningful perspective* on the role and importance of object orientation at the outset. Consequently, in Chapter 1, we present a general discussion of software engineering issues with a comparison of how the procedural and object-oriented paradigms deal with the complexity of large-scale programming projects. In later chapters, we introduce students to objects gradually, by showing them how to use some powerful, predefined classes when the need for these classes arises and is pedagogically appropriate. (These classes are discussed in detail later in this preface.)

Treatment of Class Declaration and Implementation

In regard to teaching about the declaration and implementation of classes, we have taken great care to avoid introducing too many syntactic and conceptual topics at one time. Further, we introduce a given topic, not merely for the sake of coverage but because the particular class under discussion provides real motivation for the topic. Our approach is to split the initial coverage of class declarations and implementations into two closely related chapters. In Chapter 18, we begin exploring important

syntactic and conceptual issues in object orientation by showing students how to write client programs based on reading the declaration sections of four different classes. By deferring the details of implementing these classes, we are able to provide more substantive examples of classes and, consequently, more effectively demonstrate the power of object orientation. Then, in Chapter 19, at which time students have had ample practice in using classes and reading their declarations, and have an appreciation for the power they provide, we discuss how to implement and modify the classes of Chapter 18. In Chapter 20, we give an informal discussion of designing with objects by describing actual problems and the kind of analysis and reasoning that an experienced programmer might apply to the problem. There are several classes in Chapter 20 that students will work with in the exercises.

Use of Standard Predefined Classes

The classes that we have targeted for students to use before any treatment of class declaration and implementation are (1) a subset of the ANSI/ISO Draft C++ Standard `String` class, which provides a far superior alternative to `char*`; (2) the built-in `ifstream`, `ofstream`, `istream`, and `ostream` classes required for file handling; (3) a subset of the ANSI/ISO Draft C++ Standard `Vector` class, which provides a safe, more powerful alternative to arrays; and (4) a `Matrix`, or two-dimensional `Vector`, class. Although we decided to use the previously mentioned classes primarily because we believe they make for the clearest and best pedagogy, our approach also strongly reflects recent trends in C++ instructional guidelines and is consistent with an emerging international standard for C++. First, using these classes provides the student with a suitable level of prepackaged programming power. Second, the `String`, `Vector`, and `Matrix` classes are specified by Educational Testing Service's Advanced Placement Computer Science (APCS) Committee for the AP Computer Science exams. (We anticipate that these APCS guidelines will have a major impact on the teaching of C++ in introductory courses at all levels.) Third, the `String` and `Vector` classes are clearly specified as part of the ANSI/ISO Draft C++ Standard and have already been implemented by several commercial C++ vendors.

Vectors Versus Arrays

Although we recommend the use of vectors as a safer, more convenient alternative to arrays, we introduce arrays and discuss their shortcomings before we cover vectors, and we stress that arrays and vectors have far more similarities than differences. Further, we give guidelines for rewriting programs that use vectors as programs that use arrays. Finally,

the implementation of the `Vector` class "over top of" arrays is developed fully later in the book so that students can eventually see the close connection between the two.

Order and Flexibility of Coverage of Topics

To accommodate a variety of different preferences, we have built a reasonable amount of flexibility into the book by including titled paragraphs that point out when it is possible, if so desired, to skip ahead to a related topic in a later chapter. A few examples are listed:

- We did not want to introduce all the selection structures at once in Chapter 4, so the `switch` statement, which is not a necessity, is not covered until Chapter 12. At the end of section Section 4.4, however, we point out that it is possible to jump ahead to Section 12.1 for an introduction to the `switch` statement.
- We feel that the single most important programming topic in a first course is functions. In C++, the taxonomy of function types and variations is extensive. We discuss this through several chapters, rather than attempt to cover too many function topics at once. In particular, we have deferred a detailed treatment of reference parameters until Chapter 9, rather than including them in Chapter 5, which covers both `void` and value-returning functions with value parameters. Section 5.7 briefly introduces reference parameters for use in data input functions, but coverage of this section is optional.
- Recursion is treated separately in Chapter 25 but introduced briefly in Section 10.7, which gives a scenario for early coverage of recursion.
- Sections 22.1 and 23.1–23.7 give a non-object-oriented treatment of pointers and linked lists and can be covered, if desired, any time after Chapter 13.
- Chapter 27 on inheritance can be covered any time after Chapter 19.

Pedagogical Approach

A common problem faced by instructors is finding a textbook that students will actually read and understand so that more class time can be devoted to clarification, elaboration, and integration of material, rather than to "covering" basic material that students should have learned from the book. Consequently, in writing *Using C++*, we have given serious thought to how students actually assimilate technical material. Our goal has been to make this book clear, concise, and focused by stripping down the discussion to what is essential, since too much explanation can be

just as bad as too little. Furthermore, we try to foster "active reading," by asking the reader, right in the body of the chapter, not just in the Exercise section, either to determine the output of a sample program fragment or to fill in one or two blank lines in a program whose purpose has been described. This self-testing approach gives readers a basis for determining whether they need to review material before progressing. It also provides confidence-building reinforcement for correct responses. Perhaps most important, it creates a framework that can facilitate the readers' active entry into the material.

Exercises

We believe that rich exercise sets are a vital feature of any introductory programming text. We provide three kinds of exercises in most chapters:

1. Short, objective "self-check" exercises that are designed to give students feedback on key concepts and to demonstrate an important aspect of the material. These exercises give students the kind of written practice that is often tested on exams.
2. Programming assignments whose length and level of difficulty range from short to moderate. More than one of these can be assigned for each chapter.
3. Longer, more difficult programming assignments that could be used for individual or group projects.

Software Provided on the PWS Web Site

1. We provide two short library files, `ourtools.h` and `myfuns.h`, that are used throughout the text. These files may be down loaded from the following URL: `http://www.pws.com/comsci.html`.
   a. `ourtools.h` provides the following:
      - *Simplified floating point output formatting* To avoid the syntactic baggage and confusing semantics of `setiosflags` and public data members of `ios` such as `ios::fixed`, etc., we provide the functions `fixed_out`, `scientific_out`, and `default_out`, which are first introduced in Chapter 3. The only one that we make extensive use of is the `fixed_out` function.
      - `vprn` This is an output stream that is used as a "virtual printer." Its use is first discussed in Section 2.5. In the early chapters preceding files and arrays, students can use this output destination to process input from the

monitor and send output, as a neatly formatted table, to another destination. (The implementation of this stream is trivial and is explained in Chapter 11.)

- *An `Assert` function* This function, specified by the APCS Committee, facilitates assertions with meaningful output messages for assertions that fail.

b. `myfuns.h` contains implementations of a small number of useful functions from early chapters. Students are encouraged to add other useful functions to this library.

2. We provide three library files, `bastring.h`, `bavector.h`, and `bamatrix.h`, which contain, respectively, our implementations of the `String`, `Vector`, and `Matrix` classes discussed earlier. (Complete documentation can be found in `bastring.doc`, `bavector.doc`, and `bamatrix.doc` in Appendix E.)

The five library files mentioned include all implementation code. This avoids the issue of separate compilation and building projects. Although these issues are important for large software projects, we believe that the small size of programs (even the longer projects) in introductory courses does not warrant the added complications of building projects for separate compilation. Instructors who prefer to split interface (`.h`) and implementation (`.cpp`) can have students do so as an exercise. Note, however, that the `Vector` and `Matrix` classes are templated and, as such, are not separately compilable by most compilers.

Many other Draft Standard compatible `String` and `Vector` classes are also available via the Internet, and we will provide relevant information via PWS's Web site (http://www.pws.com/comsci.html). We anticipate that these classes, if not the entire Draft Standard, will already be included in many C++ compliers when the book goes to press.

Acknowledgments

We are very indebted to Eric Bach, Indira Malik, and Ray Morin for suggesting revisions and for their help with proofreading and exercise solutions.

We would also like to express our appreciation to the following reviewers whose comments and criticisms helped shape the book:

- Don Bailes, *East Tennessee State University*
- Manuel E. Bermudez, *University of Florida*
- George Converse, *Southern Oregon State College*
- Ken Collier, *Northern Arizona University*

此为试读,需要完整PDF请访问: www.ertongbook.com

- Charles Dierbach, *Towson State University*
- H. E. Dunsmore, *Purdue University*
- Mohamed Y. Eltoweissy, *University of Pittsburgh–Johnstown*
- Susan L. Keenan, *Columbus State University*
- Anil Kini, *Texas A & M University*
- Thomas Kisko, *University of Florida*
- Daniel Ling, *Okanagan University College*
- Bonnie MacKellar, *Western Connecticut State University*
- John S. Mallozzi, *Iona College*
- Jeff McKinstry, *Point Loma Nazarene College*
- John Motil, *California State University–Northridge*
- Jean-Claude Ngatchou, *Jersey City State College*
- Rayno Niemi, *Rochester Institute of Technology*
- Ingrid Russell, *University of Hartford*
- Janet M. Urlaub, *Sinclair Community College*
- David C. Wallace, *Illinois State University*
- Wayne Wallace, *University of Wisconsin–Oshkosh*

In addition, we would like to thank the following people at PWS: Mike Sugarman, Executive Editor, for bringing about our collaboration; David Dietz, our editor, for his superb job in guiding this project; and Andrea Goldman for her excellent work as production editor.

Chuck Burchard
Julien Hennefeld

Contents

此为试读,需要完整PDF请访问: www.ertongbook.com