BRIEF EDITION

Programming and Problem Solving with C++

FIFTH EDITION

Nell Dale
University of Texas, Austin

Chip Weems
University of Massachusetts, Amherst

JONES AND BARTLETT PUBLISHERS
Sudbury, Massachusetts
BOSTON TORONTO LONDON SINGAPORE

World Headquarters
Jones and Bartlett Publishers
40 Tall Pine Drive
Sudbury, MA 01776
978-443-5000
info@jbpub.com
www.jbpub.com

Jones and Bartlett Publishers
Canada
6339 Ormindale Way
Mississauga, Ontario L5V 1J2
Canada

Jones and Bartlett Publishers
International
Barb House, Barb Mews
London W6 7PA
United Kingdom

Jones and Bartlett's books and products are available through most bookstores and online booksellers. To contact Jones and Bartlett Publishers directly, call 800-832-0034, fax 978-443-8000, or visit our website www.jbpub.com.

Substantial discounts on bulk quantities of Jones and Bartlett's publications are available to corporations, professional associations, and other qualified organizations. For details and specific discount information, contact the special sales department at Jones and Bartlett via the above contact information or send an email to specialsales@jbpub.com.

Production Credits
Publisher: David Pallai
Acquisitions Editor: Timothy Anderson
Editorial Assistant: Melissa Potter
Production Director: Amy Rose
Production Editor: Katherine Macdonald
Senior Marketing Manager: Andrea DeFronzo
V.P., Manufacturing and Inventory Control: Therese Connell
Text Design: Anne Spencer
Composition: Northeast Compositors, Inc.
Cover and Title Page Design: Kristin E. Parker
Interior Images: George Nichols
Cover and Title Page Image: © Zhang Lei/Dreamstime.com
Printing and Binding: Malloy, Inc.
Cover Printing: Malloy, Inc.

Library of Congress Cataloging-in-Publication Data
Dale, Nell B.
Programming and problem solving with C++ / Nell Dale and Chip Weems. — Brief ed.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-7637-7151-5 (pbk.)
ISBN-10: 0-7637-7151-1 (pbk.)
1. C++ (Computer program language) I. Weems, Chip. II. Title.

QA76.73.C153D34 2009
005.13'3—dc22
2008040812

6048

Printed in the United States of America
13 12 11 10 09 10 9 8 7 6 5 4 3 2 1

Contents

To quote Mephistopheles, one of the chief devils, and tempter of Faust,

...My friend, I shall be pedagogic,
And say you ought to start with Logic...
...Days will be spent to let you know
That what you once did at one blow,
Like eating and drinking so easy and free,
Can only be done with One, Two, Three.
Yet the web of thought has no such creases
And is more like a weaver's masterpieces;
One step, a thousand threads arise,
Hither and thither shoots each shuttle,
The threads flow on, unseen and subtle,
Each blow effects a thousand ties.
The philosopher comes with analysis
And proves it had to be like this;
The first was so, the second so,
And hence the third and fourth was so,
And were not the first and second here,
Then the third and fourth could never appear.
That is what all the students believe,
But they have never learned to weave.

J. W. von Goeth, *Faust*, Walter Kaufman trans., New York, 1963, 199.

As you study this book, do not let the logic of algorithms bind your imagination, but rather make it your tool for weaving masterpieces of thought.

Preface

Introduction to the Brief, Fifth Edition

The first four editions of *Programming and Problem Solving with C++* have consistently been among the best-selling computer science textbooks in the United States. These editions, as well as the Java, Ada, and Pascal versions, have been accepted widely as model textbooks for ACM/IEEE-recommended curricula for the CS1/C101 course, and for the Advanced Placement A exam in computer science.

Throughout the successive editions of this book, one thing has not changed: our commitment to the student. As always, our efforts are directed toward making the sometimes difficult concepts of computer science more accessible to all students. This edition of *Programming and Problem Solving with C++* continues to reflect our philosophy that a textbook should be like a guide, blazing a trail and leading its readers through territory that can initially seem difficult to navigate.

Changes to the Fifth Edition

We have designed this brief version of our *Programming and Problem Solving with C++, Fifth Edition*, to include only what instructors and students are able to cover in a single term. Based on the research of Elliot Soloway with novice programmers, we continue to initially cover selection using only the If statement, and loops using only the While statement. However, because many instructors like to cover all selection control structures together and all looping control structures together, we have moved the chapter on additional control structures so that it is directly after the chapters on selection and looping.

Classes and object-oriented terminology, originally in Chapter 10 with other user-defined data types, have been moved to Chapter 12, following the presentation of arrays. With this reorganization, we can go into more depth on abstract data types and the class construct used to implement them. In addition, we discuss the hallmarks of good class design.

Recognizing that many students learn programming from mimicking existing solutions, we have added numerous short example programs in every chapter. These programs illustrate chapter concepts in a more complete context than code segments, and appear immediately after the introduction of new concepts. We have also reorganized several of the chapters so that the discussion moves more quickly from a concept to its practical application early in the chapter, before moving on to related concepts. Chapters thus offer a series of concrete examples that serve as intermediate waypoints on the path to the major case studies.

Software Maintenance Case Study

Because most real-world software engineering involves working with existing code, we have added a new feature, the Software Maintenance Case Study, which demonstrates how to read code in order to debug, alter, and/or enhance an existing application or class. Although the case studies are cast in terms of revising legacy code, we have found that these skills, which are often neglected in introductory texts, are an important contributing factor to student success in writing new code.

Problem-Solving Case Study

Each chapter continues to provide a case study that illustrates algorithmic problem solving while modeling good programming practices. Each begins with a problem statement, walks through the design process, translates the design into code, and ends with a tested program. Several of the chapters have new case studies to reflect changes in chapter content.

C++ and Object-Oriented Programming

Some educators reject C and C++ as too permissive and too conducive to writing cryptic, unreadable programs. Our experience does not support this view, *provided that the use of language features is modeled appropriately*. The fact that the C family permits a terse, compact programming style cannot be labeled simply as "good" or "bad." Almost any programming language can be used to write in a style that is too terse and clever to be easily understood. The C family indeed may be used in this manner more often than are other languages, but we have found that with careful instruction in software engineering, and a programming style that is straightforward, disciplined, and free of intricate language features, students can learn to use C++ to produce clear, readable code.

It must be emphasized that although we use C++ as a vehicle for teaching computer science concepts, the book is not a language manual and does not attempt to cover all of C++. The language constructs are introduced in parallel with the appropriate theory. Thus many constructs, such as advanced object-oriented features, are not covered in this brief edition.

There are diverse opinions about when to introduce the topic of object-oriented programming (OOP). Some educators advocate an immersion in OOP from the very beginning, whereas others (for whom this book is intended) favor a more heterogeneous approach, in which both functional decomposition and object-oriented design are presented as design tools. The chapter organization of *Programming and Problem Solving with C++, Brief Edition,* reflects a transitional approach to OOP. Classes and object-oriented terminology are presented, but object-oriented design (OOD) is covered only for simple, immutable, classes. We leave a fuller treatment of OOD for the *Comprehensive Edition* of this text.

Synopsis

Chapter 1 is designed to create a comfortable rapport between students and the subject. The basics of hardware and software are presented, issues in computer ethics are raised,

C++ syntax is first encountered in a software maintenance case study, and problem-solving techniques are introduced and reinforced in a problem-solving case study.

Instead of overwhelming the student right away with the various numeric types available in C++, Chapter 2 concentrates on only two types: `char` and `string`. (For the latter, we use the ISO/ANSI string class provided by the standard library.) With fewer data types to keep track of, students can focus on overall program structure and get an earlier start on creating and running a simple program. Chapter 3 follows with a discussion of the C++ numeric types and proceeds with material on arithmetic expressions, function calls, and output. Unlike many books that detail *all* of the C++ data types and *all* of the C++ operators at once, these two chapters focus on only the `int`, `float`, `char`, and `string` types, and the basic arithmetic operators. Other data types are postponed until Chapter 10.

Input and programming methodology are the major topics of Chapter 4. The distinction between OOD and functional decomposition is explained, and the functional decomposition methodology is then presented in more depth. Students thus gain the perspective early that there are two—not just one—design methodologies in widespread use and that each serves a specific purpose. Chapter 4 also covers file I/O. The early introduction of files permits the assignment of programming problems that require the use of sample data files.

Chapter 5 begins with the concept of flow of control and branching before moving into relational and Boolean operations. Selection, using the If-Then and If-Then-Else structures, are then used to demonstrate the distinction between physical ordering of statements and logical ordering. We also develop the concept of nested control structures. Chapter 5 concludes with a lengthy Testing and Debugging section that expands on the modular design discussion by introducing preconditions and postconditions. The algorithm walk-through and code walk-through are introduced as a means of preventing errors, and the execution trace is used to find errors that may have made it into the code. We also cover data validation and testing strategies extensively in this section.

Chapter 6 is devoted to loop control strategies and looping operations using the syntax of the While statement. Rather than introducing multiple syntactical structures, our approach is to teach the concepts of looping using only the While statement. Chapter 7 covers the remaining "ice cream and cake" control structures in C++ (Switch, Do-While, and For), along with the Break and Continue statements. These structures are helpful but not essential. The section on additional C++ operators has been moved into this chapter, as they are also useful but not indispensable.

By Chapter 8, students are already comfortable with breaking problems into modules and using library functions, and they are receptive to the idea of writing their own functions. Thus Chapter 8 focuses on passing arguments by value and covers flow of control in function calls, arguments and parameters, local variables, and interface design. Coverage of interface design includes preconditions and postconditions in the interface documentation, control abstraction, encapsulation, and physical versus conceptual hiding of an implementation. Chapter 9 expands the discussion to include value-returning functions, reference parameters, scope and lifetime, stubs and drivers, and more on interface design, including side effects.

Chapter 10 begins the transition between the control structure orientation of the first part of the book and the data structure orientation of the second part. We revisit the built-in simple data types in terms of the set of values represented by each type and the allowable operations on those values. Enumeration types, structs, and unions are covered. Chapter 10 includes a discussion of simple versus structured data types.

In Chapter 11, the array is introduced as a homogeneous data structure whose components are accessed by position rather than by name. One-dimensional arrays are examined in depth, including arrays of structs. Material on two-dimensional arrays, dimensional arrays,

and multidimensional arrays rounds out the discussion of the array type. The chapter then concludes with an introduction to searching and sorting.

Chapter 12 formalizes the concept of an abstract data type as an introduction to the discussion of the class construct. Object-oriented terminology is presented, emphasizing the distinction between a class and an object. Good class design principles are stressed. The use of specification files and implementation files is presented as a form of information hiding.

Chapter 13 concludes the text with the coverage of recursion.

Additional Features

Special Sections

Five kinds of features are set off from the main text. Theoretical Foundations sections present material related to the fundamental theory behind various branches of computer science. Software Engineering Tips discuss methods of making programs more reliable, robust, or efficient. Matters of Style address stylistic issues in the coding of programs. Background Information sections explore side issues that enhance the student's general knowledge of computer science. May We Introduce sections contain biographies of computing pioneers such as Blaise Pascal, Ada Lovelace, and Grace Murray Hopper.

Goals

Each chapter begins with a list of goals for the student, broken into two categories: knowledge goals and skill goals. They are reinforced and tested in the end-of-chapter exercises.

Demonstration Programs

Much shorter and simpler than the case study examples, demonstration programs provide a bridge between syntactic concepts and their application in a problem-solving context. Each chapter now includes multiple complete demonstration programs, interspersed with coverage of new programming and language topics. All of these are available on the CD and from the web site so that students can easily experiment with them and reuse the code in their own projects.

Software Maintenance Case Studies

The majority of modern software engineering involves maintaining legacy code. It is thus essential that students learn the skills associated with reading, understanding, extending, and fixing existing programs. Such skills are rarely taught in an introductory course, where the focus tends to be on writing new programs from problem specifications. However, it turns out that these same maintenance skills are an important aspect of successfully writing new programs, because once a modest amount of code has been written, getting it to work correctly is at its essence synonymous with maintenance. These new case studies are intended to build the skills of reading, dissecting, modifying, and testing existing code.

Problem-Solving Case Studies

Problem solving is best demonstrated through case studies. In each case study, we present a problem and use problem-solving techniques to develop a manual solution. Next, we expand the solution to an algorithm, using functional decomposition, object-oriented design, or both; then we code the algorithm in C++. We show sample test data and output and follow up with a discussion of what is involved in thoroughly testing the program.

Testing and Debugging

Testing and debugging sections follow the case studies in each chapter and consider in depth the implications of the chapter material with regard to thorough testing of programs. These sections conclude with a list of testing and debugging hints.

Quick Checks
At the end of each chapter are questions that test the student's recall of major points associated with the chapter goals. Upon reading each question, the student immediately should know the answer, which he or she can then verify by glancing at the answers at the end of the section. The page number on which the concept is discussed appears at the end of each question so that the student can review the material in the event of an incorrect response.

Exam Preparation Exercises
These questions help the student prepare for tests. The questions usually have objective answers and are designed to be answerable with a few minutes of work.

Programming Warm-Up Exercises
This section provides the student with experience in writing C++ code fragments. The student can practice the syntactic constructs in each chapter without the burden of writing a complete program.

Programming Problems
These exercises, drawn from a wide range of disciplines, require the student to design solutions and write complete programs. Some of the problems are carried through multiple chapters, asking the students to reimplement the solution using new constructs or techniques, as a way of illustrating that one problem can be solved with many different approaches.

Case Study Follow-Up
These exercises give the student an opportunity to strengthen software maintenance skills by answering questions that require reading the case study code or making changes to it.

Supplements

Instructor's Resources
The online resources are powerful teaching aids available to adopters upon request from the publisher. Resources include a complete set of exercise answers, a computerized test bank, PowerPoint lecture presentations, and the complete programs from the text.

Programs
The programs contain the source code for all of the complete programs that are included within the textbook. They are available as a free download for instructors and students from the publisher's website. The programs from all of the case studies, plus complete programs that appear in the chapter bodies, are included. The program files can be viewed or edited using any standard text editor, but a C++ compiler must be used in order to compile and run the programs.

Companion Website
This website features the complete programs from the text, and the text's Appendices. Appendices A and B can also be found in the back of the text.

A Laboratory Course in C++, Fifth Edition
This lab manual follows the organization of this edition of the text. The lab manual is designed to allow the instructor maximum flexibility and may be used in both open and closed laboratory settings. Each chapter contains three types of activities: Prelab, Inlab, and Postlab. Each lesson is broken into lessons that thoroughly demonstrate the concepts covered in the

corresponding chapter. The programs, program shells (partial programs), and data files that accompany the lab manual can be found on the website for this book.

Acknowledgments

We would like to thank the many individuals who have helped us in the preparation of this fifth edition. We are indebted to the members of the faculties of the Computer Science Departments at the University of Texas at Austin and the University of Massachusetts at Amherst.

We extend special thanks to Jeff Brumfield for developing the syntax template metalanguage and allowing us to use it in the text.

For their many helpful suggestions, we thank the lecturers, teaching assistants, consultants, and student proctors who run the courses for which this book was written, as well as the students themselves.

We are grateful to the following people who took the time to offer their comments on potential changes for previous editions: Trudee Bremer, Illinois Central College; Mira Carlson, Northeastern Illinois University; Kevin Daimi, University of Detroit, Mercy; Bruce Elenbogen, University of Michigan, Dearborn; Sandria Kerr, Winston-Salem State University; Alicia Kime, Fairmont State College; Shahadat Kowuser, University of Texas, Pan America; Bruce Maxim, University of Michigan, Dearborn; William McQuain, Virginia Tech; Xiannong Meng, University of Texas, Pan America; William Minervini, Broward University; Janet Remen, Washtenaw Community College; Viviana Sandor, Oakland University; Mehdi Setareh, Virginia Tech; Katy Snyder, University of Detroit, Mercy; Tom Steiner, University of Michigan, Dearborn; John Weaver, West Chester University; Charles Welty, University of Southern Maine; Cheer-Sun Yang, West Chester University.

And thank you to the following reviewers who offered their comments for this edition: Ziya Arnavut, SUNY Fredonia; Sue Kavli, Dallas Baptist University; Katherine Snyder, University of Detroit, Mercy; Ilga Higbee, Black Hawk College; and Letha Etzkorn, University of Alabama – Huntsville.

We also thank the many people at Jones and Bartlett who contributed so much, especially Tim Anderson, Acquisitions Editor; Melissa Potter, Editorial Assistant; Katherine Macdonald, Production Editor; and Amy Rose, Production Director.

Anyone who has ever written a book—or is related to someone who has—can appreciate the amount of time involved in such a project. To our families—all of the Dale clan and the extended Dale family (too numerous to name), and to Lisa, Charlie, and Abby— thanks for your tremendous support and indulgence.

N. D.
C. W.