



BRIEF EDITION

PROGRAMMING AND PROBLEM SOLVING WITH

FIFTH EDITION

C++

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@jpub.com
www.jpub.com

Jones and Bartlett Publishers
Canada
6339 Ormendale 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.jpub.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@jpub.com.

Copyright © 2010 by Jones and Bartlett Publishers, LLC

All rights reserved. No part of the material protected by this copyright may be reproduced or utilized in any form, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without written permission from the copyright owner.

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

Preface xv

1 Overview of Programming and Problem Solving 1

- 1.1 Overview of Programming 2
 - What Is Programming? 2
 - How Do We Write a Program? 3
 - What Is an Algorithm? 4
 - What Is a Programming Language? 5
- 1.2 How Does a Computer Run a Program? 9
 - What Kinds of Instructions Can Be Written in a Programming Language? 13
 - What Is Software Maintenance? 13
- 1.3 What's Inside the Computer? 19
- 1.4 Ethics and Responsibilities in the Computing Profession 24
 - Software Piracy 24
 - Privacy of Data 25
 - Use of Computer Resources 25
 - Software Engineering 26
- 1.5 Problem-Solving Techniques 27
 - Ask Questions 27
 - Look for Things That Are Familiar 27
 - Solve by Analogy 28
 - Means-Ends Analysis 29
 - Divide and Conquer 29
 - The Building-Block Approach 29
 - Merging Solutions 30
 - Mental Blocks: The Fear of Starting 31



Algorithmic Problem Solving	31
Problem-Solving Case Study: Leap Year Algorithm	33
Summary	36
Quick Check	37
Answers	38
Exam Preparation Exercises	38
Programming Warm-Up Exercises	40
Case Study Follow-Up	41
Line Number	42

2 C++ Syntax and Semantics and the Program Development Process 43

2.1	The Elements of C++ Programs	44
	C++ Program Structure	44
	Syntax and Semantics	46
	Syntax Templates	47
	Naming Program Elements: Identifiers	49
	Data and Data Types	50
	Naming Elements: Declarations	53
	Taking Action: Executable Statements	57
	Beyond Minimalism: Adding Comments to a Program	62
2.2	Program Construction	63
	Blocks (Compound Statements)	65
	The C++ Preprocessor	67
	Software Maintenance Case Study: Adding Titles to Names	69
2.3	More about Output	72
	Creating Blank Lines	72
	Inserting Blanks within a Line	73
	Special Characters	74
2.4	Program Entry, Correction, and Execution	75
	Entering a Program	75
	Compiling and Running a Program	75
	Problem-Solving Case Study: Printing a Chessboard	76
	Testing and Debugging	80
	Summary	81
	Quick Check	81
	Answers	82
	Exam Preparation Exercises	83
	Programming Warm-Up Exercises	85



Programming Problems 86
Case Study Follow-Up 87

3 Numeric Types, Expressions, and Output 89

- 3.1 Overview of C++ Data Types 90
- 3.2 Numeric Data Types 90
 - Integral Types 90
 - Floating-Point Types 92
- 3.3 Declarations for Numeric Types 93
 - Named Constant Declarations 93
 - Variable Declarations 94
- 3.4 Simple Arithmetic Expressions 95
 - Arithmetic Operators 95
 - Increment and Decrement Operators 97
- 3.5 Compound Arithmetic Expressions 98
 - Precedence Rules 98
 - Type Coercion and Type Casting 99
- Software Maintenance Case Study:** Precedence Error 102
- 3.6 Function Calls and Library Functions 104
 - Value-Returning Functions 104
 - Library Functions 106
 - Void Functions 108
- 3.7 Formatting Output 108
 - Integers and Strings 109
 - Floating-Point Numbers 111
- 3.8 Additional `string` Operations 117
 - The `length` and `size` Functions 117
 - The `find` Function 118
 - The `substr` Function 120
 - Accessing Characters Within a String: The `at` Function 121
 - Converting to Lowercase and Uppercase 121
- Problem-Solving Case Study:** Mortgage Payment Calculator 124
- Testing and Debugging 127
- Summary 127
- Quick Check 128
- Answers 129
- Exam Preparation Exercises 129
- Programming Warm-Up Exercises 131
- Programming Problems 132
- Case Study Follow-Up 133



4 Program Input and the Software Design Process 135

4.1	Getting Data into Programs	136
	Input Streams and the Extraction Operator (<code>>></code>)	137
	The Reading Marker and the Newline Character	140
	Reading Character Data with the <code>get</code> Function	141
	Skipping Characters with the <code>ignore</code> Function	144
	Reading String Data	145
4.2	Interactive Input/Output	147
4.3	Noninteractive Input/Output	149
4.4	File Input and Output	150
	Files	150
	Using Files	150
	Software Maintenance Case Study: Adding File Input/Output to a Program	154
	Run-Time Input of File Names	157
4.5	Input Failure	159
4.6	Software Design Methodologies	160
4.7	Functional Decomposition	161
	Modules	162
	Implementing the Design	163
	A Perspective on Design	167
	Problem-Solving Case Study: Displaying a Name in Multiple Formats	168
	Testing and Debugging	172
	Testing and Debugging Hints	173
	Summary	174
	Quick Check	175
	Answers	175
	Exam Preparation Exercises	175
	Programming Warm-Up Exercises	178
	Programming Problems	179
	Case Study Follow-Up	181

5 Conditions, Logical Expressions, and Selection Control Structures 183

5.1	Flow of Control	184
	Selection	184
5.2	Conditions and Logical Expressions	185
	The <code>bool</code> Data Type	185
	Logical Expressions	186
5.3	The If Statement	190



The If-Then-Else Form	190
Blocks (Compound Statements)	193
The If-Then Form	196
A Common Mistake	197
Software Maintenance Case Study: Incorrect Output	198
5.4 Nested If Statements	201
The Dangling <code>else</code>	204
5.5 Logical Operators	205
Precedence of Operators	211
Relational Operators with Floating-Point Types	213
5.6 Testing the State of an I/O Stream	213
Problem-Solving Case Study: BMI Calculator	215
Testing and Debugging	221
Testing in the Problem-Solving Phase: The Algorithm Walk-Through	221
Testing in the Implementation Phase	223
The Test Plan	227
Tests Performed Automatically During Compilation and Execution	228
Testing and Debugging Hints	229
Summary	231
Quick Check	231
Answers	231
Exam Preparation Exercises	232
Programming Warm-Up Exercises	234
Programming Problems	236
Case Study Follow-Up	238

6 Looping 239

6.1 The While Statement	240
6.2 Phases of Loop Execution	242
6.3 Loops Using the While Statement	242
Count-Controlled Loops	243
Event-Controlled Loops	245
Looping Subtasks	251
Software Maintenance Case Study: Make a Program General	255
6.4 How to Design Loops	259
Designing the Flow of Control	259
Designing the Process Within the Loop	261
The Loop Exit	261
6.5 Nested Logic	262
Designing Nested Loops	266
Problem-Solving Case Study: Recording Studio Design	273



Testing and Debugging	284
Loop-Testing Strategy	284
Test Plans Involving Loops	284
Testing and Debugging Hints	285
Summary	287
Quick Check	287
Answers	288
Exam Preparation Exercises	289
Programming Warm-Up Exercises	291
Programming Problems	292
Case Study Follow-Up	295

7 Additional Control Structures 297

7.1	The Switch Statement	298
7.2	The Do-While Statement	304
7.3	The For Statement	308
Software Maintenance Case Study: Changing a Loop Implementation 312		
7.4	The Break and Continue Statements	314
7.5	Guidelines for Choosing a Looping Statement	316
7.6	Additional C++ Operators	316
	Assignment Operators and Assignment Expressions	318
	Increment and Decrement Operators	318
	Bitwise Operators	319
	The Case Operation	320
	The <code>sizeof</code> Operator	320
	The <code>? :</code> Operator	321
	Operator Precedence	322
	Type Coercion in Arithmetic and Relational Expressions	323
Problem-Solving Case Study: The Rich Uncle 325		
Testing and Debugging 329		
	Testing and Debugging Hints	329
	Summary	330
	Quick Check	330
	Answers	331
	Exam Preparation Exercises	331
	Programming Warm-Up Exercises	333
	Programming Problems	334
	Case Study Follow-Up	337



8 Functions 339

8.1	Functional Decomposition with Void Functions	340
	When to Use Functions	340
	Why Do Modules Need an Interface Design?	341
	Designing Interfaces	341
	Writing Modules as Void Functions	343
8.2	An Overview of User-Defined Functions	347
	Flow of Control in Function Calls	347
	Function Parameters	347
8.3	Syntax and Semantics of Void Functions	350
	Function Call (Invocation)	350
	Function Declarations and Definitions	351
	Local Variables	353
	The Return Statement	354
8.4	Parameters	355
	Value Parameters	356
	Reference Parameters	357
	Software Maintenance Case Study: Refactoring a Program	359
	Using Expressions with Parameters	365
	A Last Word of Caution About Argument and Parameter Lists	366
	Writing Assertions as Function Documentation	368
	Problem-Solving Case Study: Lawn Care Company Billing	371
	Testing and Debugging	381
	The <code>assert</code> Library Function	381
	Testing and Debugging Hints	383
	Summary	384
	Quick Check	384
	Answers	385
	Exam Preparation Exercises	385
	Programming Warm-Up Exercises	387
	Programming Problems	388
	Case Study Follow-Up	392

9 Scope, Lifetime, and More on Functions 393

9.1	Scope of Identifiers	394
	Scope Rules	396
	Variable Declarations and Definitions	399
	Namespaces	400



9.2	Lifetime of a Variable	402
	Initializations in Declarations	403
	Software Maintenance Case Study: Debug a Simple Program	404
9.3	Interface Design	408
	Side Effects	408
	Global Constants	411
9.4	Value-Returning Functions	413
	Complete Example	416
	Boolean Functions	420
	Interface Design and Side Effects	422
	When to Use Value-Returning Functions	423
9.5	Type Coercion in Assignments, Argument Passing, and Return of a Function Value	424
	Problem-Solving Case Study: Health Profile	427
	Testing and Debugging	436
	Stubs and Drivers	437
	Testing and Debugging Hints	441
	Summary	442
	Quick Check	443
	Answers	443
	Exam Preparation Exercises	444
	Programming Warm-Up Exercises	446
	Programming Problems	447
	Case Study Follow-Up	450

10 User-Defined Data Types 451

10.1	Built-In Simple Types	452
	Numeric Types	453
	Characters	454
10.2	User-Defined Simple Types	456
	The Typedef Statement	456
	Enumeration Types	457
	Named and Anonymous Data Types	465
10.3	Simple Versus Structured Data Types	466
10.4	Records (Structs)	467
	Accessing Individual Components	470
	Aggregate Operations on Structs	471
	More About Struct Declarations	473
	Binding Like Items	474
	Software Maintenance Case Study: Changing a Loop Implementation	474



10.5	Hierarchical Records	476
10.6	Unions	478
	Problem-Solving Case Study: Stylistical Analysis of Text	480
	Testing and Debugging	488
	Coping with Input Errors	488
	Testing and Debugging Hints	488
	Summary	488
	Quick Check	489
	Answers	490
	Exam Preparation Exercises	490
	Programming Warm-Up Exercises	492
	Programming Problems	492
	Case Study Follow-Up	495

11 Arrays 497

11.1	One-Dimensional Arrays	498
	Declaring Arrays	500
	Accessing Individual Components	501
	Out-of-Bounds Array Indexes	503
	Initializing Arrays in Declarations	505
	(Lack of) Aggregate Array Operations	505
	Examples of Declaring and Accessing Arrays	506
	Passing Arrays as Arguments	512
	Commenting Arrays	515
	Software Maintenance Case Study: Modularizing a Program	516
	Using Typedef with Arrays	519
11.2	Arrays of Records	520
	Arrays of Records	520
11.3	Special Kinds of Array Processing	523
	Subarray Processing	523
	Indexes with Semantic Content	526
11.4	Two-Dimensional Arrays	526
11.5	Passing Two-Dimensional Arrays as Arguments	530
11.6	Processing Two-Dimensional Arrays	532
	Sum the Rows	532
	Sum the Columns Revised	533
	Sum the Columns	535
	Initialize the Array	535
	Print the Array	536



11.7	Another Way of Defining Two-Dimensional Arrays	539
11.8	Multidimensional Arrays	541
11.9	Sorting and Searching in an Array	543
	Sorting	543
	Searching	547
	Problem-Solving Case Study: Calculating Exam Statistics	550
	Problem-Solving Case Study: Favorite Rock Group	558
	Testing and Debugging	566
	One-Dimensional Arrays	566
	Complex Structures	566
	Multidimensional Arrays	568
	Sorting and Searching	568
	Testing and Debugging Hints	569
	Summary	570
	Quick Check	570
	Answers	571
	Exam Preparation Exercises	571
	Programming Warm-Up Exercises	574
	Programming Problems	576
	Case Study Follow-Up	578
12	Classes and Abstraction	579
12.1	Abstract Data Types	580
12.2	C++ Classes	583
	Implementing the Member Functions	588
	Classes, Objects, and Members	591
	Built-in Operations on Objects	592
	Class Scope	593
12.3	Information Hiding	594
	User-Written Header Files	595
	Specification and Implementation Files	596
	Compiling and Linking a Multifile Program	602
12.4	What Is an Object?	603
12.5	Class Design Principles	607
	Encapsulation	607
	Abstraction	609
	Designing for Modifiability and Reuse	610
	Mutability	611
	Software Maintenance Case Study: Comparing Two <code>TimeOfDay</code> Objects	613



12.6	The Name ADT	618
	Specification of the ADT	619
	Implementation File	621
12.7	Composition	623
	Design of an Entry Class	623
12.8	UML Diagrams	628
	Diagramming a Class	628
	Diagramming Composition of Classes	629
	Problem-Solving Case Study: Create an Array of Name Objects	629
	Testing and Debugging	636
	Testing and Debugging Hints	639
	Summary	640
	Quick Check	641
	Answers	641
	Exam Preparation Exercises	641
	Programming Warm-Up Exercises	642
	Programming Problems	644
	Case Study Follow-Up	646

13 Recursion 647

13.1	What Is Recursion?	648
13.2	Recursive Algorithms with Simple Variables	651
13.3	Towers of Hanoi	653
13.4	Recursive Algorithms with Structured Variables	657
13.5	Recursion or Iteration?	660
	Software Maintenance Case Study: Substituting Binary Search for Linear Search	660
	Testing and Debugging	665
	Testing and Debugging Hints	665
	Summary	665
	Quick Check	665
	Answers	666
	Exam Preparation Exercises	666
	Programming Warm-Up Exercises	668
	Programming Problems	670
	Case Study Follow-Up	671

Index 673



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; Xian-nong 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.