大学计算机教育丛书（影印版）

C++ PROGRAM DESIGN

An Introduction to Programming and Object-Oriented Design

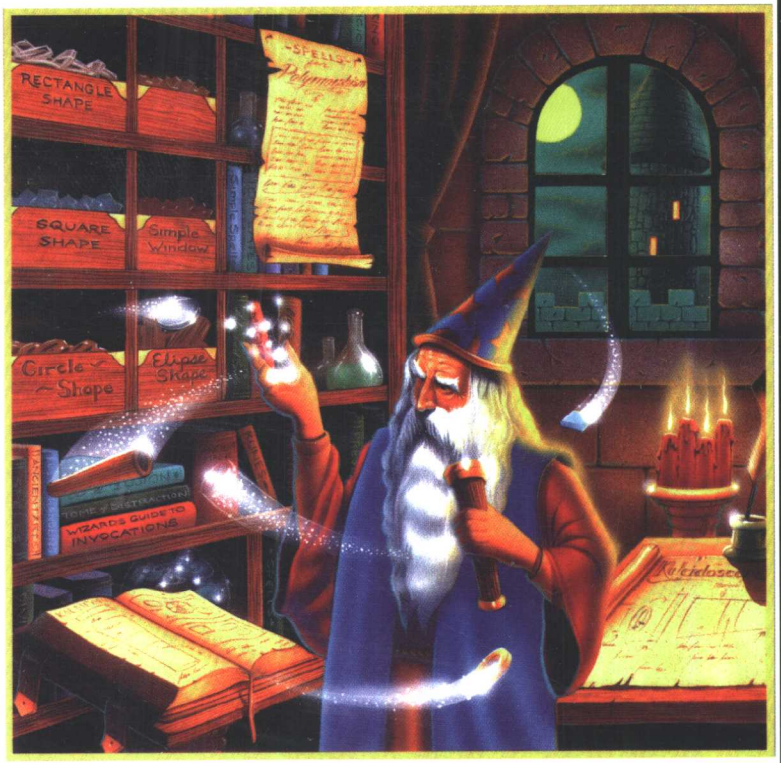
COHOON & DAVIDSON

3rd Edition

C++程序设计

程序设计和面向对象设计入门

第3版



清华大学出版社
http://www.tup.tsinghua.edu.cn

McGraw-Hill
http://www.mhhe.com

C++ Program Design

An Introduction to Programming and Object-Oriented Design

Third Edition

C++程序设计

程序设计和面向对象设计入门

第 3 版

James P. Cohoon
University of Virginia

Jack W. Davidson
University of Virginia

清华大学出版社

McGraw-Hill Companies, Inc.

（京）新登字 158 号

C++ PROGRAM DESIGN: AN INTRODUCTION TO PROGRAMMING
AND OBJECT-ORIENTED DESIGN, THIRD EDITION
Cohoon, James P. /Davidson, Jack W.

北京市版权局著作权合同登记号：图字：01-2001-4316

书　名：C++程序设计——程序设计和面向对象设计入门（第 3 版）
作　者：James P. Cohoon　Jack W. Davidson
出版者：清华大学出版社（北京清华大学学研大厦，邮编 100084）
　　http:// www.tup.tsinghua.edu.cn
印刷者：清华大学印刷厂
发行者：新华书店总店北京发行所
开　本：787×960 1/16　印张：61.5
版　次：2002 年 1 月第 1 版　2002 年 1 月第 1 次印刷
书　号：ISBN 7-900637-50-8
印　数：0001～3000
定　价：76.00 元（含光盘）

出 版 者 的 话

今天，我们的大学生、研究生和教学、科研工作者，面临的是一个国际化的信息时代。他们将需要随时查阅大量的外文资料；会有更多的机会参加国际性学术交流活动；接待外国学者；走上国际会议的讲坛。作为科技工作者，他们不仅应有与国外同行进行口头和书面交流的能力，更为重要的是，他们必须具备极强的查阅外文资料获取信息的能力。有鉴于此，在原国家教委所颁布的“大学英语教学大纲”中有一条规定：专业阅读应作为必修课程开设。同时，在大纲中还规定了这门课程的学时和教学要求。有些高校除开设“专业阅读”课之外，还在某些专业课拟进行英语授课。但教、学双方都苦于没有一定数量的合适的英文原版教材作为教学参考书。为满足这方面的需要，我们陆续精选了一批国外计算机科学方面最新版本的著名教材，进行影印出版。我社获得国外著名出版公司和原著作者的授权将国际先进水平的教材引入我国高等学校，为师生们提供了教学用书，相信会对高校教材改革产生积极的影响。

我们欢迎高校师生将使用影印版教材的效果、意见反馈给我们，更欢迎国内专家、教授积极向我社推荐国外优秀计算机教育教材，以利于我们将《大学计算机教育丛书（影印版）》做得更好，更适合高校师生的需要。

清华大学出版社

《大学计算机教育丛书（影印版）》项目组

1999.6

Preface

INTRODUCTION

Computers are an inescapable fixture in our lives. They control complex systems such as financial networks, mass transit, telephone systems, and power plants. Tens of millions of people use the Internet to access information, shop, recreate, communicate, and conduct business. Because the computer has become such an intrinsic component of modern life, we believe that everyone should have a basic working knowledge of how computers are programmed.

This textbook is about the fundamentals of programming and software development using C++, a popular high-level programming language developed by Bjarne Stroustrup of AT&T Bell Laboratories. We chose C++ for teaching programming because it supports the development of software using the object-oriented approach. An advantage of object-oriented development is that it lets us build complex software systems employing many of the techniques that have been used for constructing complex physical systems, such as cars, airplanes, or buildings. This book is targeted for a first programming course, and it has been designed to be appropriate for people from all disciplines. We assume no prior programming skills and use mathematics and science at a level appropriate to first-year college students.

Some of this book's important features are

- *The C++ standard is given broad coverage.* Our original naive intent was to offer complete coverage of C++. However, such a presentation would be overwhelming for the beginning student. For example, the language definition describes more than 150 standard classes and libraries. Rather than being encyclopedic, we provide in-depth coverage of all materials that an introductory course would need, introduce much of the remaining material, and give pointers to the rest. We also provide integrated coverage of the important additions and modifications to the C++ language, such as type **bool**, Standard Template Library, namespaces, and exceptions. The breadth of our coverage provides flexibility for the instructor. For example, an instructor may choose not to cover inheritance, but instead to cover

templates. For the students, the coverage allows advanced learners to go further in the language, and it makes the book valuable as a reference source.

- *Classes are introduced early.* Chapter 1 includes a gentle introduction to the object-oriented paradigm. Material is presented there to whet students' appetites. We believe that students must first be client users of objects before they can appreciate the difficulties of designing flexible, usable objects. All proficient designers started as users. The next several chapters introduce and use some standard stream class objects, such as `cout` and `cin`, `string` objects using the Standard Template Library, and a limited number of objects derived from a graphical library developed for the textbook. This experience helps reinforce the concepts of encapsulation, software reuse, and the object-oriented programming paradigm. After this solid introduction to the use of objects, we present approximately 50 classes and ADTs over the final eight chapters.
- *We present the use of a graphical Application Programmer Interface (API) designed specifically for beginning programmers to develop interesting programs.* We provide a portable, object-oriented graphical library, named EzWindows, for the easy display of simple geometric, bitmap, and text objects. We supply implementations of the API for popular Windows and UNIX compilers. Using the API provides several important experiences for the student. First, students are client users of a software library. As mentioned earlier, using well-designed objects helps novice programmers begin to appreciate good object-oriented design. Their experience as users forms the basis for becoming designers. Second, using the API introduces students to the real-world practice of developing programs using an application-specific library. Third, using EzWindows to perform graphical input and output exposes the student to event-based programming and the dominant mode of input and output used in real applications, and it permits development of exciting and visually interesting programs. This experience motivates the students, and it provides a visually concrete set of objects that help students understand the object-oriented paradigm. EzWindows is simple enough to allow even the first programming assignments to be graphical. Examples using EzWindows are sprinkled throughout the text. However, the presentation is done in a fashion that accommodates instructors who prefer to cover only ANSI materials. For students and instructors who want more advanced graphic capabilities, see Appendix E, which is a complete EzWindows reference. For sample assignments and supplementary materials, visit our Web sites `http://www.mhhe.com/c++programdesign` and `http://www.cplusplus-programdesign.com`.
- *Software-engineering design concepts are introduced via problem studies and software projects.* Besides containing numerous small examples for introducing C++ and object-oriented design concepts, each chapter considers one or more problems in detail. As appropriate, there are object-oriented analysis and design, algorithm development, and code to realize

the design. In addition, two chapters are devoted to the principles of software project development using our EzWindows API (Chapters 10 and 15). These chapters are springboards for software reuse and for projects suitable for individual and group work.

- *Programming and style tips are presented in boxes that clearly separate this material from the main text.* In addition to explaining C++ and object-oriented programming, we also give advice on how to be a better and more knowledgeable programmer and designer. For example, there are important tips on avoiding common programming errors, writing readable code, and understanding the new directions the standard took, as well as tips on performance and software engineering. Boxes also present one or two pages per chapter of historical information on computing.
- *Integrated use of the Standard Template Library.* An important component of the C++ language is its Standard Template Library, or as it is more commonly known, the STL. This library provides a rich collection of container classes for representing lists and strings and a set of generic algorithms for important programming tasks such as sorting, searching, and list traversal. Readers first encounter the STL in Chapter 3 when we introduce the string class. In subsequent chapters, additional features are introduced. In particular, Chapter 9 thoroughly explores list representation using the STL's vector container class, and Chapters 11 and 14 give insight on how the STL container classes can be implemented and use their generic algorithms to solve several common programming tasks.

THIRD EDITION HIGHLIGHTS

This third edition incorporates many of the suggestions we have received from both instructors and students. Some of the notable improvements and additions in this edition are:

- *Self-check questions.* At appropriate points in each chapter there are self-check questions that students can use to check their mastery of the material. The self-check questions are in addition to the exercises at the end of the chapter. The solutions to the self-check questions are available at our Web sites `www.mhhe.com/c++programdesign` and `www.cplusplus-programdesign.com`. The self-check questions include both short answer questions as well as programming exercises.
- *Earlier coverage of classes.* The material on classes is now covered a chapter earlier (Chapter 7). This change reflects the growing consensus that early coverage of classes is possible and the right approach for teaching object-oriented programming using C++.
- *Coverage of testing and debugging.* An important skill for programmers is how to test and debug the programs they write. Previous editions of the text did not cover this important topic. Chapter 12 provides an introduction to testing and debugging. The chapter discusses various testing techniques such as unit testing, integration testing, and code inspections. The sections

on debugging focus on teaching students how to use the scientific method to find bugs. The chapter also discusses common bugs encountered by beginning programmers and how to recognize them.

- *Improved explanations.* The entire text has been reexamined and, based on our own analysis and user feedback, additional examples have been inserted, clarifying figures have been created, and, as appropriate, explanations have been revised and expanded.

CONTEXT

In the early 1990s, with the support of the National Science Foundation, the Department of Computer Science at the University of Virginia began developing a new computer science curriculum. We carefully examined our existing curriculum and those of several other peer schools.

What we found were curricula that emphasized the following:

- Use of a programming language that is rarely used outside of undergraduate courses.
- Construction of small programs, consisting of at most a few hundred lines.
- Development in isolation of text-based programs "from scratch" for each assignment.
- Development in an environment lacking modern tools.
- An informal development with the belief that if a program "works," it is acceptable.

Comparing this situation with the real world, we saw considerable differences. Practicing computer professionals:

- Use programming languages designed for developing large applications that are often thousands or even millions of source lines long.
- Are involved most often in modifying and maintaining such systems rather than in developing them.
- Work in teams, not as single programmers.
- Do system development according to mandated specifications.
- Build systems that use graphical user interfaces to do input and output.
- Use existing libraries and tools to build systems.

To better prepare our students for real-world programming, we developed the first edition of this book. This third edition reflects feedback we have received from both instructors and students as well as our own experience using the book in large introductory programming courses and in a second course on programming and software engineering.

Programming

Most of the important concepts and problems in computer science cannot be appreciated unless one has a good understanding of what a program is and how to write one. Unfortunately, learning to program is difficult. Programming

well, like writing well, takes years of practice. In fact, teaching programming and teaching writing are, in some respects, very similar.

Students are taught writing by reading examples of good prose and by writing, writing, writing. In the process, they learn the important skill of how to organize ideas so they can be presented effectively. As students develop their skills, they move from writing and editing a paragraph or several paragraphs to creating larger pieces of prose, such as essays, short stories, and reports.

Our approach to teaching programming is similar to teaching writing but with one very important addition. Throughout the text, we present and discuss many examples of both good and bad programming. Programming exercises give the student the opportunity to practice organizing and writing code. In addition, we offer examples that facilitate learning the practical skill of modifying existing code. This is done through the use of code that is specifically designed to be modified by the student. We have found this mechanism to be effective because it forces the student to read and understand the provided code. In the text, a CD-ROM/World Wide Web icon signals that this code is available on the CD-ROM included with the book and at our Web site.

Why C++?

As we began our new curriculum development, one of the first issues was choosing which programming language to use. Like many departments, we had been using Pascal. Although it was unanimously decided that Pascal should be replaced, the choice of a replacement was the subject of much heated debate. Some of the languages we considered were C, C++, Modula-3, Scheme, and Smalltalk. A deciding factor was that we wanted to use a language that we ourselves use professionally. This decision narrowed the choices to C or C++. Although the decision was not unanimous, we chose C++ based on the belief that the object-oriented paradigm would be the dominant programming paradigm of the future.

In hindsight, we made the correct choice. C++ has continued to grow in popularity, and many companies use it as their development language. Indeed, many of our graduates report that when they interview for a job, a question they are often asked is whether they know C++. We believe that we will see a continuing shift to C++ as the introductory programming language of choice. We have also been pleasantly surprised by the effect on our students. The students in our upper-level courses who have completed our software development sequence can tackle much larger and harder problems than the students who had completed the comparable sequence in our old curriculum. In addition, we have seen substantial migration of other disciplines to C++. For example, all students in the commerce school at our university now take C++, and the engineering disciplines that had previously required Fortran now require C++.

Some of you may be wondering about Java. Java is definitely an interesting language. However, serious software development tools are not yet mature, and the language and its libraries are still undergoing serious revisions. The conventional wisdom of the professionals in our research areas is that Java

might be the language for developing graphical user interfaces (GUIs) but that C++ remains the language for application development. If we are to meet our goal of most ably preparing students for computing careers, then C++ is the right educational vehicle.

Introduce objects early

Our experience of teaching C++ over the past 8 years shows that the object-oriented paradigm can be successfully introduced to beginning programmers. In our initial course offerings, we introduced objects near the end of the course and did superficial coverage of objects, classes, overloading, and inheritance. Essentially, we taught C using C++ syntax and input and output mechanisms. This approach failed. It introduced a new concept too late in the course—students were not able to integrate the material. We revised our course to introduce objects earlier and found that this approach worked much better. Students now have time to absorb this material because it is used and reinforced throughout the course rather than just at the end. The objects-early approach is reflected in this text. Students begin using standard objects in Chapter 2. Chapters 3 through 7 introduce the students to the use of graphical objects from the EzWindows API. After this solid introduction to using objects, Chapter 7 introduces classes and the design of objects, and it logically follows the chapters that introduce control structures, functions, and libraries (Chapters 4 through 6). We strongly believe that this is the proper sequencing of the material in an introductory textbook. The students certainly like it! By their second and third assignment they are producing useful software with graphic capabilities.

Software projects

As noted, what we had been teaching in the past was not at all close to what was happening in the real world. To educate future computer scientists in the skills that support the engineering and comprehension of large software systems, reengineering of existing systems, and application of innovative techniques (such as software reuse), our department deemed it necessary to begin introducing this material in the first course. Our software project chapters (Chapters 10 and 15) are vehicles for this introduction. These chapters provide several important experiences for the student. First, both projects use our EzWindows API. Using the API to do event-based programming and graphical input and output exposes students to the programming model typically used in real-world applications, and it permits students to develop more exciting and interesting programs. If desired, the software projects facilitate students' working together in groups of up to four. Again, this practice mirrors the real world, where it is rare for a lone programmer to develop an application. The software projects also illustrate software maintenance. Many of the exercises at the end of the software project chapters call for the student to make major modifications or nontrivial extensions to the project program.

CHAPTER SUMMARY

- *Chapter 1: Computing and the object-oriented design methodology*—basic computing terminology, machine organization, software, software development, software engineering, object-oriented design and programming.
- *Chapter 2: C++: the fundamentals*—program organization, function main(), include statement, comments, definitions, writing readable code, interactive input and output, fundamental types, literals, constants, declarations, expressions, conversions, precedence.
- *Chapter 3: Modifying objects*—assignment statement and conversions, extractions, const objects, increment and decrement, insertion and extractions, string class, Standard Template Library, graphical objects and the EzWindows API.
- *Chapter 4: Control constructs*—logical values and operators, truth tables, bool, relational operators, general precedence, short-circuit evaluation, if statement, if-else statement, sorting, switch statement, enum, while statement, for statement, invariants, do statement, text processing, scientific visualization.
- *Chapter 5: Function usage basics and libraries*—functions, value parameters, formal parameters, actual parameters, invocation, flow of control, activation records, pseudorandom numbers, prototyping, preprocessor, inclusion directives, header files, conditional compilation, software reuse, using libraries, standard streams, manipulators, file streams, file processing, iostream, iomanip, fstream, ctype, string, stdlib, and assert libraries.
- *Chapter 6: Programmer-defined functions*—function definitions, parameters, invocation, flow of control, return statement, scope, local objects, global objects, reference parameters, constant parameters, default parameters, parameter casting, function overloading, initialization, name reuse, top-down design, recursion, in-memory streams, utility functions, Standard Template Library, integrating a quadratic polynomial, financial visualization.
- *Chapter 7: The class construct and object-oriented design*—programmer-defined data types, class construct, information hiding, encapsulation, object-oriented analysis and design, access specification, data members, member functions, constructors, kaleidoscope program, object-oriented factory automation simulator/trainer.
- *Chapter 8: Implementing abstract data types*—data abstraction, object-oriented design, default and copy constructors, inspectors, mutators, facilitators, auxiliary functions, memberwise assignment, const member functions, arithmetic operator overloading, reference return, insertion and extraction overloading, pseudorandom-number generation, ADTs for rational and pseudorandom numbers, and the red-yellow-green game.

- *Chapter 9: Lists*—one-dimensional arrays, subscripting, parameter passing, initialization, character strings, multidimensional lists, tables, matrices, Standard Template Library, container classes, adapter classes, vector class, vector member functions, sorting, InsertionSort, QuickSort, binary search, two-dimensional search, list representation, initialization lists, iterators, ADTs for maze-traversing robot problem.

- *Chapter 10: The EzWindows API: a detailed examination*—Application Programmer Interfaces, graphical user interface, event-based programming, window coordinate system, callbacks, mouse and timer events, EzWindows API mechanics, ADTs for simple windows, bitmaps, text labels, and a Simon Says game.

- *Chapter 11: Pointers and dynamic memory*—lvalues, rvalues, pointer types, addressing, indirection, pointers as parameters, pointers to pointers, constant pointers, equivalence of array and pointer notation, character string processing, command-line parameters, pointers to functions, dynamic objects, free store, new and delete operators, dangling pointers, memory leak, destructors, member assignment, this, ADT for a list of integers.

- *Chapter 12: Testing and debugging*—black-box testing, white-box testing, inspections, unit testing, integration testing, system testing, statement coverage, equivalence partitioning, regression testing, boundary conditions, code reviews, test harness, path coverage.

- *Chapter 13: Inheritance*—object-oriented design, reuse, base class, derived class, single inheritance, is-a relationship, has-a relationship, uses-a relationship, shape hierarchy, controlling inheritance, protected members, multiple inheritance, ADTs for rectangles, circles, ellipses, and triangles, an object-oriented kaleidoscope program.

- *Chapter 14: Templates and polymorphism*—generic actions and types, function template, class template, container class, sequential lists, linked list, iterator class, friends, polymorphism, virtual function, pure virtual function, abstract base class, virtually derived class, virtual multiple inheritance, list ADTs, random-access list, sequential lists, list iterators, singly linked lists, doubly linked lists.

- *Chapter 15: Software project—bug hunt!*—encapsulation, inheritance, virtual functions, object-oriented design, Bug Hunt game, ADTs for various kinds of bugs and a game controller.

- *Appendixes* — ASCII character set, general precedence table, iostream, stdlib, time, string and algorithm libraries, vector and other container classes, string class, namespaces, using statements, exceptions, friends, EzWindows API, project and make files.

USING THIS BOOK

This text has more material than can be covered in a single course. The extra coverage was deliberate—it allows instructors to select their choice of topics on programming and software development. The book was also designed for flexibility in teaching. For example, if an instructor desires to move the introduction of classes earlier in the course, he or she can cover iteration after classes and our development of the rational number ADT. If an instructor desires to introduce classes after arrays, then Sections 9.1 to 9.5 and Section 9.12 of Chapter 9 can precede Chapters 7 and 8. Also, the discussion of inheritance in Chapter 13 can precede the coverage of pointers and dynamic objects in Chapter 11. Instructors who do a breadth-first coverage of computer science may choose to omit the software project chapters and substitute material from sources that cover topics such as the social and ethical aspects of computing or elementary formal logic. The testing material of Chapter 12 (Section 12.1) can be covered anytime after the material on classes (Chapter 7) has been introduced. The section on debugging (Section 12.2) relies on array examples, and therefore should be covered after Chapters 9 and 11.

We use the following layout for our course.

Week	Topic	Readings
1	Computing and object-oriented design	Chapter 1
2	Programming fundamentals	Chapter 2
3	Object manipulation	Chapter 3
4	Conditional statements	Chapter 4 (Sections 4.1–4.6)
5	Iteration statements	Chapter 4 (Sections 4.7–4.12)
6	Functions and reuse	Chapter 5, Chapter 6
7	Parameter passing	Chapter 6
8	OO analysis and design	Chapter 7
9–10	ADTs	Chapter 8 (Sections 8.1–8.6)
11	Arrays	Chapter 9 (Sections 9.1–9.4, Section 9.12)
12	Vectors	Chapter 9 (Sections 9.5–9.10)
13	Project—Simon Says, OOA/OOD	Chapter 10
14	Inheritance	Chapter 13

Depending on faculty interests, the material covered in week 13 can vary. In the introductory course at our university, we spend one week every semester on a problem in detail. Generally, this examination contributes to the final project.

SUPPLEMENTARY MATERIALS

In addition to the included CD-ROM, which contains source code and supplementary files for many of our programs and listings, we have developed other

materials. For example, there is a set of slide transparencies (approximately 300 slides). The course we teach also has a closed-laboratory component that meets once a week for reinforcing current course topics. For these laboratories, we have developed a student laboratory manual. These materials are available from the publisher. For more detailed information, visit our Web site at `http://www.mhhe.com/c++programdesign`. In particular, we maintain a frequently asked question list (FAQ) and links to helpful C++ and educational sites. Other educational supplements are also available at our class Web site `http://www.cs.virginia.edu/cs101`.

SYMBOLS

The following icons are used in the margins throughout the text.

The World Wide Web (WWW) and CD-ROM icon is associated with some code listings and programs. This icon indicates that the code is available both on the CD-ROM supplied with the book and at our Web site `http://www.mhhe.com/c++programdesign`. When the icon is associated with the label Program, the program consists of a single file. If the icon is associated with the label Listing, a library file or one file in a multifile program is being made available.

The exclamation icon indicates a warning about programming. Often these are tips on how to avoid common programming errors.

The detour icon indicates a set of self-check exercises. The answers to the exercises can be found at our Web site `www.mhhe.com/c++programdesign` or `www.cplusplus-programdesign.com`. The self-check exercises include both short answer questions as well as programming exercises.

The sunglass icon indicates that the associated material is related to programming style. At the current time, a number of conventions are being used. The manner that code is presented in this text generally reflects the dominant convention. (Of course, our variation is the best!)

The book icon indicates that the associated material is concerned with the C++ programming language itself. The two typical uses of this icon are for advanced C++ topics or for describing a recent language extension that can have an impact on software development.

The spotlight icon indicates programming tips or highlights material that presents a more detailed discussion or a sidebar to the current topic.

The abacus icon indicates a discussion on the history of computing. Many people often mistakenly think that computing is simply writing programs. While designing and writing programs is certainly an important part of computing, it is by no means the only thing encompassed by computing. Each chapter contains at least one anecdote regarding triumphs and failures of the pioneers in computing.

THE AUTHORS

Jim Cohoon is a professor in the computer science department at the University of Virginia and is a former member of the technical staff at AT&T Bell Laboratories. He joined the faculty after receiving his Ph.D. from the University of Minnesota. He has been nominated twice by the department for the university's best-teaching award. In 1994, Professor Cohoon was awarded a Fulbright Fellowship to Germany, where he lectured on C++ and software engineering. Professor Cohoon's research interests include algorithms, computer-aided design of electronic systems, optimization strategies, and computer science education. He is the author of more than 60 papers in these fields. He is a member of the Association of Computing Machinery (ACM), the ACM Special Interest Group on Design Automation (SIGDA), the ACM Special Interest Group on Computer Science Education (SIGCSE), the Institute of Electrical and Electronics Engineers (IEEE), and the IEEE Circuits and Systems Society. He is a member of the ACM Publications and SIG Boards and is past chair of SIGDA. He can be reached at `cohoon@virginia.edu`. His Web homepage is `http://www.cs.virginia.edu/~cohoon`.

Jack Davidson is also a professor in the computer science department at the University of Virginia. He joined the faculty after receiving his Ph.D. from the University of Arizona. Professor Davidson has received NCR's Faculty Innovation Award for innovation in teaching. Professor Davidson's research interests include compilers, computer architecture, systems software, and computer science education. He is the author of more than 80 papers in these fields. He is a member of the ACM, the ACM Special Interest Group on Programming Languages (SIGPLAN), the ACM Special Interest Group on Computer Architecture (SIGARCH), SIGCSE, the IEEE, and the IEEE Computer Society. He served as an associate editor of *Transactions on Programming Languages and Systems*, ACM's flagship journal on programming languages and systems, from 1994 to 2000. He was chair of the 1998 Programming Language Design and Implementation Conference (PLDI '98) and program co-chair of the 2000 SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems

(LCTES 2000). He can be reached at jwd@virginia.edu. His Web homepage is http://www.cs.virginia.edu/~jwd.

DELVING FURTHER

The following are primary references on the C++ language.

- International Standard for Information Systems—Programming Language C++, ISO/IEC FDIS 14882, Washington, DC: American National Standards Institute, 1998.
- B. Stroustrup, *The C++ Programming Language,* 3rd ed., Reading, MA: Addison-Wesley, 1998.

The following are good sources on libraries and more-advanced object-oriented design, program development, and the Standard Template Library.

- J. Bergin, *Data Abstraction: The Object-Oriented Approach Using C++*, New York: McGraw-Hill, 1994.
- M. D. Carroll and M. A. Ellis, *Designing and Coding Reusable C++*, Reading, MA: Addison-Wesley, 1995.
- M. P. Cline and G. A. Lomow, *C++ FAQs*, Reading, MA: Addison-Wesley, 1995.
- A. Koenig and B. Moo, *Ruminations on C++*, Reading, MA: Addison-Wesley, 1997.
- S. B. Lippman and J. Lajoie, *C++ Primer,* 3rd ed., Reading, MA: Addison-Wesley, 1998.
- S. Maguire, *Writing Solid Code*, Redmond, WA: Microsoft Press, 1993.
- S. Meyers, *Effective C++*, Reading, MA: Addison-Wesley, 1998.
- S. Meyers, *More Effective C++*, Reading, MA: Addison-Wesley, 1996.
- D. R. Musser and A. Saini, *STL Tutorial and Reference Guide*, Reading, MA: Addison-Wesley, 1995.
- P. J. Plauger, A. Stepanov, M. Lee, and D. R. Musser, *The Standard Template Library*, Englewood Cliffs, NJ: Prentice-Hall, 1998.
- B. Stroustrup, *The Design and Evolution of C++*, Reading, MA: Addison-Wesley, 1994.

The following are good sources for learning more about the history and future of computing.

- S. Augarten, *Bit by Bit: An Illustrated History of Computers*, New York: Ticknor & Fields, 1984.
- P. J. Denning and B. Metcalfe (eds.), *Beyond Calculation: The Next Fifty Years of Computing*, New York: Copernicus Press, Springer-Verlag, 1997.
- J. A. N. Lee, *Computer Pioneers*, Piscattaway, NJ: IEEE Press, 1995.
- J. Palfreman and D. Swade, *The Dream Machine: Exploring the Computer Age*, London: BBC Books, 1991.

- H. G. Stine, *The Untold Story of the Computer Revolution*, New York: Arbor House, 1985.
- M. R. Williams, *A History of Computing Technology*, Englewood Cliffs, NJ: Prentice-Hall, 1985.

ACKNOWLEDGMENTS

We thank the University of Virginia for providing an environment that made this book possible. In particular, we thank Jack Stankovic for his tireless efforts in leading the computer science department to national prominence. We also thank Mark Bailey, Alan Batson, Joanne Cohoon, Clark Coleman, John Karro, Sean McCulloch, James Ortega, Jane Prey, Paul Reynolds, and Alfred Weaver for their comments. Thanks also goes to Bruce Childers who helped design and implement the original EzWindows API and Peter Valle who helped revise the EzWindows API for this edition.

We are grateful to Rich Rashid and Amitabh Srivastava of Microsoft Research for providing an environment that allowed the third edition to be completed. Microsoft Research is a great place to think, write, and do research. A very special thanks go to David Hanson, Todd Proebsting, and Chris Fraser of the Programming Language Systems group at Microsoft Research for making us feel welcome and at home. We will miss the interesting and stimulating lunch-time conversations—especially the lunch trips to Hole-in-the-Wall Barbeque for Meatloaf Monday and to Acapalco Fresh for Burrito Thursday.

We thank all of the people at McGraw-Hill for their efforts in making this edition a reality. In particular, we thank Tom Casson, for his support and encouragement; Kay Brimeyer, for her behind-the-scenes product-management skills; John Wannemacher, for his creative marketing ideas; David Hash, for leading the cover-design team; June Waldman, for copyediting the second edition; Jill Barrie for copyediting the third edition; and Janelle Pregler for her careful proofreading of the third edition. Special thanks go again to Elizabeth (Betsy) Jones, our executive editor, for support, direction, and focus throughout this project, and Kelley Butcher, our senior developmental editor, for managing and synthesizing the reviewing process.

We thank the following class testers, readers, and reviewers for their valuable comments and suggestions on the second edition of this text:

Kenneth Bayse, Clark University
Leslie Blackford, Wheaton College
Jacobo Carrasquel, Carnegie Mellon University
John Dailey Jr., University of Illinois
Suzanne Miller Dorney, Grand Valley State University
Gerald Dueck, Brandon University
H. E. Dunsmore, Purdue University
Elizabeth Lee Falta, Louisiana Tech University
William Filter, Sandia National Laboratories
Ann Ford, University of Michigan