Preface

Welcome to ANSI/ISO Standard C, and to C++ and Java™, too! At Deitel & Associates, we write college-level programming-language textbooks and professional books and work hard to keep our published books up-to-date with a steady flow of new editions. Writing *C How to Program, Fourth Edition*, (*4/e* for short), was a joy. This book and its support materials have everything instructors and students need for an informative, interesting, challenging and entertaining educational experience. We have tuned the writing, the pedagogy, our coding style and the book's ancillary package. Also, we have included a Tour of the Book in this Preface. This will help instructors, students and professionals get a sense of the rich coverage this book provides of C, C++ and Java programming.

In this Preface, we overview the conventions we use in *C How to Program, 4/e*, such as syntax coloring the code examples, "code washing" and highlighting important code segments to help focus students' attention on key concepts introduced in each chapter. We also overview the new features of *C How to Program, 4/e*.

Prentice Hall has bundled Microsoft's Visual C++® 6 Introductory Edition software with the text. To further support novice programmers, we offer several of our new *Dive-Into™ Series* publications that are available free for download at `www.deitel.com`. These materials explain how to compile, execute and debug C, C++ and Java programs using various popular development environments.

We discuss *C How to Program, 4/e*'s comprehensive suite of educational materials that help instructors maximize their students' learning experience. These include an Instructor's Resource CD with solutions to the book's chapter exercises and a Test-Item File with hundreds of multiple-choice examination questions and answers. Additional instructor resources are available at the book's Companion Web Site (`www.prenhall.com/deitel`), which includes a Syllabus Manager and customizable PowerPoint® Lecture Notes. PowerPoint slides and additional support materials are available for students at the Companion Web Site, as well.

We also discuss the new *Student Solutions Manual* to accompany this textbook. This additional resource provides solutions to about half of the exercises in the textbook.

C How to Program, 4/e, was reviewed by a team of distinguished academics and industry professionals, including the head and former head of the C standards committee; we list their names and affiliations so you can get a sense of how carefully this book was scrutinized. The Preface concludes with information about the authors and about Deitel & Associates, Inc. As you read this book, if you have any questions, please send an e-mail to `deitel@deitel.com`; we will respond promptly. Please visit our Web site, `www.deitel.com`, regularly and be sure to sign up for the *Deitel® Buzz Online* e-mail newsletter at `www.deitel.com/newsletter/subscribe.html`. We use the Web site and the newsletter to keep our readers current on all Deitel publications and services.

Features of C How to Program, Fourth Edition

Syntax Highlighting

We syntax highlight all the C, C++ and Java code, as do many integrated-development environments and code editors. This greatly improves code readability—an especially important goal, given that this book contains over 13,280 lines of code. Our syntax-highlighting conventions are as follows:

```
comments appear like this
keywords appear like this
errors appear like this
constants and literal values appear like this
all other code appears in black
```

Code Highlighting and User-Input Highlighting

We have added extensive code highlighting to make it easier for readers to spot the featured segments of each program. This feature also helps students review the material rapidly when preparing for exams or labs. We have also highlighted in our screen dialogs all user inputs to distinguish them from program outputs.

"Code Washing"

"Code washing" is our term for applying comments, using meaningful identifiers, applying indentation and using vertical spacing to separate meaningful program units. This process results in programs that are much more readable and self-documenting. We have added extensive and descriptive comments to all of the code, including a comment before and after every major control statement, to help the student clearly understand the flow of the program. We have done extensive code washing of all the source code programs in the text, the ancillaries and the *Student Solutions Manual*.

To promote good programming practices, we updated all the source code programs in the C portion of this book with new coding standards. Variable definitions are now placed on separate lines to increase readability and every control statement has an opening and closing brace even when this is redundant. This will help the reader when he or she is beginning to develop large and complex programs. Every function prototype now matches the first line of the function definition, including the parameter names (which help document the program and reduce errors—especially for novice programmers).

Use of Terminology/Presentation
We have updated our use of terminology throughout the text to comply with the various language standards and specification.

Teaching Approach

Many educators believe that the complexity of C, and a number of other difficulties, make C unworthy for a first programming course—precisely the target course for this book. So why did we write this text?

Dr. Harvey M. Deitel (HMD) taught introductory programming courses in college environments for two decades with an emphasis on developing clearly written, well-structured programs. Much of what is taught in these courses is the basic principles of structured programming, with an emphasis on the effective use of control statements and functionalization. We have presented this material exactly the way HMD has done in his college courses. Students are motivated by the fact that they are learning a language that will be immediately useful to them as they enter industry.

Our goal was clear: Produce a C programming textbook for introductory university-level courses in computer programming for students with little or no programming experience, yet offer the deep and rigorous treatment of theory and practice demanded by traditional C courses. To meet these goals, we produced a book larger than other C texts—this because our text also patiently teaches structured programming principles. Hundreds of thousands of students worldwide have learned C from the earlier editions of this book.

C How to Program, 4/e, contains a rich collection of examples, exercises and projects drawn from many fields and designed to provide students with a chance to solve interesting, real-world problems. The code examples in the text have been tested on multiple compilers.

The book concentrates on the principles of good software engineering and stresses program clarity. We are educators who teach edge-of-the-practice topics in industry classrooms worldwide. This text emphasizes good pedagogy.

Live-Code Approach
C How to Program, 4/e, is loaded with numerous live-code examples— each new concept is presented in the context of a complete, working program that is immediately followed by one or more sample executions showing the program's input/output dialog. This style exemplifies the way we teach and write about programming. We call this method of teaching and writing the live-code approach. *We use programming languages to teach programming languages.* Reading the examples in the text is much like typing and running them on a computer.

World Wide Web Access
All of the source-code examples for *C How to Program, 4/e,* (and our other publications) are available on the Internet as downloads from the following Web sites:

```
www.deitel.com
www.prenhall.com/deitel
```

Registration is quick and easy and the downloads are free. We suggest downloading all the examples, then running each program as you read the corresponding text. Making changes to the examples and immediately seeing the effects of those changes is a great way to enhance your learning experience.

Objectives
Each chapter begins with a statement of objectives. This tells the student what to expect and gives the student an opportunity, after reading the chapter, to determine if he or she has met these objectives. It is a confidence builder and a source of positive reinforcement.

Quotations
The learning objectives are followed by a series of quotations. Some are humorous, some are philosophical and some offer interesting insights. Our students enjoy relating the quotations to the chapter material. You may appreciate some of the quotations more *after* reading the chapters.

Outline
The chapter outline helps the student approach the material in top-down fashion. This, too, helps students anticipate what is to come and set a comfortable and effective learning pace.

Sections
Each chapter is organized into small sections that address key C, C++ or Java topics.

13, 280 Lines of Syntax-Highlighted Code in 268 Example Programs (with Outputs)
We present C, C++ and Java features in the context of complete, working programs using our our live-code approach. Each program is immediately followed by a window containing the outputs produced when the program is run. This enables the student to confirm that the programs run as expected. Relating outputs to the program statements that produce those outputs is an excellent way to learn and to reinforce concepts. Our programs exercise many features of C, C++ and Java. Reading the book carefully is much like entering and running these programs on a computer. The code is "syntax highlighted" with keywords appearing in bold blue, comments appearing in italic blue, constants and literal values appearing in a lighter shade of bold blue and the rest of each program appearing in black. This makes it much easier to read the code—students will especially appreciate the syntax highlighting when they read the more substantial programs we present.

469 Illustrations/Figures
An abundance of colorized charts and line drawings is included. The discussions of control statements in Chapters 3 and 4 feature carefully drawn flowcharts. [*Note:* We do not teach the use of flowcharting as a program development tool, but we do use a brief flowchart-oriented presentation to specify the precise operation of C's control statements.] Chapter 12, Data Structures, uses colorized line drawings to illustrate creating and maintaining linked lists, queues, stacks and binary trees. The remainder of the book is abundantly illustrated.

768 Programming Tips
We have included seven programming tip elements to help students focus on important aspects of program development, testing and debugging, performance and portability. We highlight hundreds of these tips in the form of *Common Programming Errors*, *Error-Prevention Tips*, *Good Programming Practices*, *Look-and-Feel Observations*, *Performance Tips*, *Portability Tips* and *Software Engineering Observations*. These tips and practices represent the best we have been able to glean from six decades (combined) of programming and teaching experience. One of our students—a mathematics major—told us that she feels this approach is like the highlighting of axioms, theorems and corollaries in mathematics books; it provides a basis on which to build good software.

259 Common Programming Errors

Students learning a language—especially in their first programming course—tend to make certain kinds of errors frequently. Focusing on these Common Programming Errors *helps students avoid making the same errors. It also helps reduce long lines outside instructors' offices during office hours*!

132 Good Programming Practices

Good Programming Practices *are tips for writing clear programs. These techniques help students produce programs that are more readable, self-documenting and easier to maintain.*

49 Error-Prevention Tips

When we first designed this "tip type," we thought we would use it strictly to tell people how to test and debug programs and in previous editions have labelled this tip as "Testing and Debugging Tips." In fact, many of the tips describe aspects of C, C++ and Java that reduce the likelihood of "bugs" and thus simplify the testing and debugging processes. In addition, we also changed many of the Good Programming Practices *throughout the book to this tip type.*

32 Look-and-Feel Observations

In the Java portion of this book, we provide Look-and-Feel Observations *to highlight graphical user interface conventions. These observations help students design their own graphical user interfaces to conform with industry norms.*

68 Performance Tips

In our experience, teaching students to write clear and understandable programs is by far the most important goal for a first programming course. But students want to write the programs that run the fastest, use the least memory, require the smallest number of keystrokes, or dazzle in other nifty ways. Students really care about performance. They want to know what they can do to "turbo charge" their programs. So we highlight opportunities for improving program performance—making programs run faster or minimizing the amount of memory that they occupy.

38 Portability Tips

Software development is a complex and expensive activity. Organizations that develop software must often produce versions customized to a variety of computers and operating systems. So there is a strong emphasis today on portability, i.e., on producing software that will run on a variety of computer systems with few, if any, changes. Many people tout C, C++ and Java as appropriate languages for developing portable software. Some people assume that if they implement an application in one of the languages, the application will automatically be portable. This is simply not the case. Achieving portability requires careful and cautious design. There are many pitfalls. We include numerous Portability Tips *to help students write portable code. Java was designed from the start to maximize portability, but Java programs can also require modifications to "port" them.*

189 Software Engineering Observations

The Software Engineering Observations *highlight techniques, architectural issues and design issues, etc. that affect the architecture and construction of software systems, especially large-scale systems. Much of what the student learns here will be useful in upper-level courses and in industry as the student begins to work with large, complex real-world systems. C, C++ and Java are especially effective software engineering languages.*

Summary
Each chapter ends with additional pedagogical devices. We present an extensive, bullet-list-style *Summary* in every chapter. This helps the student review and reinforce key concepts. There is an average of 37 summary bullets per chapter.

Terminology
We include a *Terminology* section with an alphabetized list of the important terms defined in the chapter—again, further reinforcement. There is an average of 73 terms per chapter.

Summary of Tips, Practices and Errors
We collect and list from the chapter the *Good Programming Practices*, *Common Programming Errors*, *Look-and-Feel Observations*, *Performance Tips*, *Portability Tips*, *Software Engineering Observations* and *Error-Prevention Tips*.

728 Self-Review Exercises and Answers (Count Includes Separate Parts)
Extensive *Self-Review Exercises* and *Answers to Self-Review Exercises* are included for self study. This gives the student a chance to build confidence with the material and prepare to attempt the regular exercises.

993 Exercises (Count Includes Separate Parts; 1722 Total Exercises)
Each chapter concludes with a substantial set of exercises including simple recall of important terminology and concepts; writing individual program statements; writing small portions of functions and C++/Java classes; writing complete functions, C++/Java classes and programs;and writing major term projects. The large number of exercises enables instructors to tailor their courses to the unique needs of their audiences and to vary course assignments each semester. Instructors can use these exercises to form homework assignments, short quizzes and major examinations.

4800+ Index Entries (Total of 7500+ Entries Counting Multiple References)
We have included an extensive *Index* at the back of the book. This helps the student find any term or concept by keyword. The *Index* is useful to people reading the book for the first time and is especially useful to practicing programmers who use the book as a reference. Most of the terms in the *Terminology* sections appear in the *Index* (along with many more index entries from each chapter). Thus, the student can use the *Index* in conjunction with the *Terminology* sections to be sure he or she has covered the key material of each chapter.

Software Included with C How to Program, 4/e

In writing this book, we have used a variety of C compilers. For the most part, the programs in the text will work on all ANSI/ISO C and C++ compilers, including the Visual C++ 6.0 Introductory Edition compiler included with this book.

The C material (Chapters 2–14) follows the ANSI C standard published in 1990. See the reference manuals for your particular system for more details about the language, or obtain a copy of ANSI/ISO 9899: 1990, “American National Standard for Information Systems—Programming Language C,” from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036.

In 1999, ISO approved a new version of C, C99, which is not as yet widely used. Appendix B contains a comprehensive list of C99 Web resources. For more information on C99—and to purchase a copy of the C99 standards document (ISO/IEC 9899:1999)—visit the Web site of the American National Standards Institute (ANSI) at `www.ansi.org`.

The C++ material is based on the C++ programming language as developed by the Accredited Standards Committee INCITS, Information Technology and its Technical Committee J11, Programming Language C++, respectively. The C and C++ languages were approved by the International Standards Organization (ISO).

The serious programmer should read these documents carefully and reference them regularly. These documents are not tutorials. Rather they define their respective languages with the extraordinary level of precision that compiler implementors and "heavy-duty" developers demand.

The Java chapters are based on Sun Microsystem's Java programming language. Sun provides an implementation of the Java 2 Platform called the Java 2 Software Development Kit (J2SDK) that includes the minimum set of tools you need to write software in Java. You can download the most recent version the J2SDK from

```
java.sun.com/j2se/downloads.html
```

Information on installing and configuring the J2SDK is located at

```
developer.java.sun.com/developer/onlineTraining/new2java/
gettingstartedjava.html
```

We have carefully audited our presentation against these documents and documentation. Our book is intended to be used at the introductory and intermediate levels. We have not attempted to cover every feature discussed in these comprehensive documents.

DIVE-INTO™ Series Tutorials for Popular C, C++ and Java Environments

We have launched our new *DIVE-INTO™ SERIES* of tutorials to help our readers get started with many popular program-development environments. These are available free for download at `www.deitel.com/books/downloads.html`.

Currently, we have the following *DIVE-INTO™ SERIES* publications:

- *DIVE-INTO Microsoft® Visual C++® 6*
- *DIVE-INTO Microsoft® Visual C++® .NET*
- *DIVE-INTO Borland™ C++Builder™ Compiler* (command-line version)
- *DIVE-INTO Borland™ C++Builder™ Personal* (IDE version)
- *DIVE-INTO GNU C++ on Linux*
- *DIVE-INTO GNU C++ via Cygwin on Windows* (Cygwin is a UNIX emulator for Windows that includes the GNU C++ compiler.)
- *DIVE-INTO Forte for Java Community Edition 3.0*
- *DIVE-INTO SunOne Studio Community Edition 4.0*

Each of these tutorials shows how to compile, execute and debug C, C++ and Java applications in that particular compiler product. Many of these documents also provide step-by-step instructions with screen shots to help readers install the software. Each document overviews the compiler and its online documentation.

Ancillary Package for C How to Program, 4/e

C How to Program, 4/e, has extensive ancillary materials for instructors. The *Instructor's Resource CD (IRCD)* contains solutions to most of the end-of-chapter exercises. This CD is available only to instructors through their Prentice Hall representatives. **[*NOTE:* Please do not write to us requesting the instructor's CD. Distribution of this CD is limited strictly to college professors teaching from the book. Instructors may obtain the solutions manual only from their Prentice Hall representatives.]** The ancillaries for this book also include a *Test Item File* of multiple-choice questions. In addition, we provide PowerPoint® slides containing all the code and figures in the text and bulleted items that summarize the key points in the text. Instructors can customize the slides. The PowerPoint® slides are downloadable from `www.deitel.com` and are available as part of Prentice Hall's Companion Web Site (`www.prenhall.com/deitel`) for *C How to Program, 4/e*, which offers resources for both instructors and students. For instructors, the Companion Web Site offers a Syllabus Manager, which helps instructors plan courses interactively and create online syllabi.

Students also benefit from the functionality of the *Companion Web Site*. Book-specific resources for students include:

- Customizable PowerPoint® slides
- Source code for all example programs
- Reference materials from the book appendices (such as operator-precedence chart, character set and Web resources)

Chapter-specific resources available for students include:

- Chapter objectives
- Highlights (e.g., chapter summary)
- Outline
- Tips (e.g., *Common Programming Errors*, *Good Programming Practices*, *Portability Tips*, *Performance Tips*, *Look-and-Feel Observations*, *Software Engineering Observations* and *Error-Prevention Tips*)
- Online Study Guide—contains additional short-answer self-review exercises (e.g., true/false and matching questions) with answers and provides immediate feedback to the student

Students can track their results and course performance on quizzes using the *Student Profile* feature, which records and manages all feedback and results from tests taken on the *Companion Web Site*. To access DEITEL® *Companion Web Site*, visit `www.prenhall.com/deitel`.

Student Solutions Manual

The *C Student Solutions Manual* (ISBN 0-13-145245-2) to accompany *C How to Program, 4/e* provides solutions to approximately half of the exercises in the text. Many of the solved exercises are similar to the unsolved exercises, which will help students when completing homework assignments.

DEITEL® e-Learning Initiatives

e-Books and Support for Wireless Devices

Wireless devices will have an enormous role in the future of the Internet. Given recent bandwidth enhancements and the emergence of 2.5 and 3G technologies, it is projected that, within a few years, more people will access the Internet through wireless devices than through desktop computers. Deitel & Associates is committed to wireless accessibility and has published *Wireless Internet & Mobile Business How to Program*. We are investigating new electronic formats, such as wireless e-books so that students and professors can access content virtually anytime, anywhere. For periodic updates on these initiatives subscribe to the *DEITEL® Buzz Online* e-mail newsletter, `www.deitel.com/newsletter/subscribe.html` or visit `www.deitel.com`.

DEITEL® Buzz Online E-mail Newsletter

Our free e-mail newsletter, the *DEITEL® Buzz Online*, includes commentary on industry trends and developments, links to free articles and resources from our published books and upcoming publications, product-release schedules, errata, challenges, anecdotes, information on our corporate instructor-led training courses and more. To subscribe, visit

```
www.deitel.com/newsletter/subscribe.html
```

The New DEITEL® Developer Series

Deitel & Associates, Inc., is making a major commitment to covering leading-edge technologies for industry software professionals through the launch of our *DEITEL® Developer Series*. The first books in the series are *Web Services A Technical Introduction* and *Java Web Services for Experienced Programmers*. We are working on *ASP .NET with Visual Basic .NET for Experienced Programmers*, *ASP .NET with C# for Experienced Programmers* and many more. Please visit `www.deitel.com` or subscribe to our e-mail newsletter at `www.deitel.com/newsletter/subscribe.html` for continuous updates on all published and forthcoming *DEITEL Developer Series* titles.

The *DEITEL Developer Series* is divided into three subseries. The *A Technical Introduction* subseries provides IT managers and developers with detailed overviews of emerging technologies. The *A Programmer's Introduction* subseries is designed to teach the fundamentals of new languages and software technologies to programmers and novices from the ground up; these books discuss programming fundamentals, followed by brief introductions to more sophisticated topics. The *For Experienced Programmers* subseries is designed for seasoned developers seeking a deeper treatment of new programming languages and technologies, without the encumbrance of introductory material; the books in this subseries move quickly to in-depth coverage of the features of the programming languages and software technologies being covered.

A Tour of the Book

The book is divided into four major parts. The first part, Chapters 1 through 14, presents a thorough treatment of the C programming language including a formal introduction to structured programming. The second part (Chapters 15 through 23)—unique among C textbooks—presents a substantial treatment of C++ and object-oriented programming sufficient for an upper-level undergraduate college course. The third part—Chapters 24 through

30 (and also unique among C books)—presents a thorough introduction to Java, including graphics programming, graphical user interface (GUI) programming using Java Swing, multimedia programming and event-driven programming. The fourth part, Appendices A through F, presents a variety of reference materials that support the main text.

Part 1: Procedural Programming in C

Chapter 1—Introduction to Computers, the Internet and the World Wide Web— discusses what computers are, how they work and how they are programmed. It introduces the notion of structured programming and explains why this set of techniques has fostered a revolution in the way programs are written. The chapter gives a brief history of the development of programming languages from machine languages, to assembly languages, to high-level languages. The origins of the C, C++ and Java programming languages are discussed. The chapter includes an introduction to a typical C programming environment. We discuss the explosion in interest in the Internet that has occurred with the advent of the World Wide Web and the Java programming language.

Chapter 2—Introduction to C Programming—gives a concise introduction to writing C programs. A detailed treatment of decision making and arithmetic operations in C is presented. After studying this chapter, the student will understand how to write simple, but complete, C programs.

Chapter 3—Structured Program Development—is probably the most important chapter in the text, especially for the serious student of computer science. It introduces the notion of algorithms (procedures) for solving problems. It explains the importance of structured programming in producing programs that are understandable, debuggable, maintainable and likely to work properly on the first try. It introduces the fundamental control statements of structured programming, namely the sequence, selection (`if` and `if…else`) and repetition (`while`) statements. It explains the technique of top-down, stepwise refinement that is critical to the production of properly structured programs. It presents the popular program design aid, structured pseudocode. The methods and approaches used in Chapter 3 are applicable to structured programming in any programming language, not just C. This chapter helps the student develop good programming habits in preparation for dealing with the more substantial programming tasks in the remainder of the text.

Chapter 4—C Program Control—refines the notions of structured programming and introduces additional control statements. It examines repetition in detail and compares the alternatives of counter-controlled loops and sentinel-controlled loops. The `for` statement is introduced as a convenient means for implementing counter-controlled loops. The `switch` selection statement and the `do…while` repetition statement are presented. The chapter concludes with a discussion of logical operators.

Chapter 5—C Functions—discusses the design and construction of program modules. C's function-related capabilities include standard library functions, programmer-defined functions, recursion and call-by-value capabilities. The techniques presented in Chapter 5 are essential to the production and appreciation of properly structured programs, especially the kinds of larger programs and software that system programmers and application programmers are likely to develop in real-world applications. The "divide and conquer" strategy is presented as an effective means for solving complex problems by dividing them into simpler interacting components. Students enjoy the treatment of random numbers and simulation, and they appreciate the discussion of the dice game of craps which makes elegant use of control statements. We introduce enumerations in this chapter and provide a

more detailed discussion in Chapter 10. Chapter 5 offers a solid introduction to recursion and includes a table summarizing the dozens of recursion examples and exercises distributed throughout the remainder of the book. Some books leave recursion for a chapter late in the book; we feel this topic is best covered gradually throughout the text. The extensive exercises include several classical recursion problems such as the Towers of Hanoi.

Chapter 6—C Arrays—discusses the structuring of data into arrays, or groups, of related data items of the same type. The chapter presents numerous examples of both single-subscripted arrays and double-subscripted arrays. It is widely recognized that structuring data properly is just as important as using control statements effectively in developing properly structured programs. The examples investigate various common array manipulations, printing histograms, sorting data, passing arrays to functions and an introduction to the field of survey data analysis (with simple statistics). A feature of this chapter is the careful discussion of elementary sorting and searching techniques and the presentation of binary searching as a dramatic improvement over linear searching. The end-of-chapter exercises include a variety of interesting and challenging problems, such as improved sorting techniques, the design of an airline reservations system, an introduction to the concept of turtle graphics (made famous in the LOGO language) and the Knight's Tour and Eight Queens problems that introduce the notions of heuristic programming so widely employed in the field of artificial intelligence.

Chapter 7—C Pointers—presents one of the most powerful and difficult to master features of the C language: pointers. The chapter provides detailed explanations of pointer operators, call by reference, pointer expressions, pointer arithmetic, the relationship between pointers and arrays, arrays of pointers and pointers to functions. The chapter exercises include a delightful simulation of the classic race between the tortoise and the hare, card shuffling and dealing algorithms and recursive maze traversals. A special section entitled "Building Your Own Computer" is also included. This section explains machine language programming and proceeds with a project involving the design and implementation of a computer simulator that allows the reader to write and run machine language programs. This unique feature of the text will be especially useful to the reader who wants to understand how computers really work. Our students enjoy this project and often implement substantial enhancements, many of which are suggested in the exercises. In Chapter 12, another special section guides the reader through building a compiler; the machine language produced by the compiler is then executed on the machine language simulator produced in Chapter 7.

Chapter 8—C Characters and Strings—deals with the fundamentals of processing nonnumeric data. The chapter includes a thorough walkthrough of the character and string processing functions available in C's libraries. The techniques discussed here are widely used in building word processors, page layout and typesetting software and text-processing applications. The chapter includes a variety of exercises that explore text-processing applications. The student will enjoy the exercises on writing limericks, writing random poetry, converting English to pig Latin, generating seven-letter words that are equivalent to a given telephone number, text justification, check protection, writing a check amount in words, generating Morse Code, metric conversions and dunning letters. The last exercise challenges the student to use a computerized dictionary to create a crossword puzzle generator.

Chapter 9—C Formatted Input/Output—presents all the powerful formatting capabilities of `printf` and `scanf`. We discuss `printf`'s output formatting capabilities such as rounding floating point values to a given number of decimal places, aligning columns of

numbers, right-justification and left-justification, insertion of literal information, forcing a plus sign, printing leading zeros, using exponential notation, using octal and hexadecimal numbers and controlling field widths and precisions. We discuss all of `printf`'s escape sequences for cursor movement, printing special characters and causing an audible alert. We examine all of `scanf`'s input formatting capabilities, including inputting specific types of data and skipping specific characters in an input stream. We discuss all of `scanf`'s conversion specifiers for reading decimal, octal, hexadecimal, floating point, character and string values. We discuss scanning inputs to match (or not match) the characters in a scan set. The chapter exercises test virtually all of C's formatted input/output capabilities.

Chapter 10—C Structures, Unions, Bit Manipulations and Enumerations—presents a variety of important features. Structures are like records in other programming languages—they group data items of various types. Structures are used in Chapter 11 to form files consisting of records of information. Structures are used in conjunction with pointers and dynamic memory allocation in Chapter 12 to form dynamic data structures such as linked lists, queues, stacks and trees. Unions enable an area of memory to be used for different types of data at different times; such sharing can reduce a program's memory requirements or secondary-storage requirements. Enumerations provide a convenient means of defining useful symbolic constants; this helps make programs more self-documenting. C's powerful bit manipulation capabilities enable programmers to write programs that exercise lower-level hardware capabilities. This helps programs process bit strings, set individual bits on or off and store information more compactly. Such capabilities, often found only in low-level assembly languages, are valued by programmers writing system software such as operating systems and networking software. A feature of the chapter is its revised, high-performance card shuffling and dealing simulation. This is an excellent opportunity for the instructor to emphasize the quality of algorithms.

Chapter 11—C File Processing—discusses the techniques used to process text files with sequential access and random access. The chapter begins with an introduction to the data hierarchy from bits, to bytes, to fields, to records, to files. Next, C's simple view of files and streams is presented. Sequential-access files are discussed using programs that show how to open and close files, how to store data sequentially in a file and how to read data sequentially from a file. Random-access files are discussed using programs that show how to create a file sequentially for random access, how to read and write data to a file with random access and how to read data sequentially from a randomly accessed file. The fourth random-access program combines many of the techniques of accessing files both sequentially and randomly into a complete transaction-processing program.

Chapter 12—C Data Structures—discusses the techniques used to create and manipulate dynamic data structures. The chapter begins with discussions of self-referential structures and dynamic memory allocation and proceeds with a discussion of how to create and maintain various dynamic data structures including linked lists, queues (or waiting lines), stacks and trees. For each type of data structure, we present complete, working programs and show sample outputs. The chapter helps the student master pointers. It includes abundant examples using indirection and double indirection—a particularly difficult concept. One problem when working with pointers is that students have trouble visualizing the data structures and how their nodes are linked together. So we have included illustrations that show the links, and the sequence in which they are created. The binary tree example is a nice capstone for the study of pointers and dynamic data structures. This example creates a

binary tree; enforces duplicate elimination; and introduces recursive preorder, inorder and postorder tree traversals. Students have a genuine sense of accomplishment when they study and implement this example. They particularly appreciate seeing that the inorder traversal prints the node values in sorted order. The chapter includes a substantial collection of exercises. A highlight of the exercises is the special section "Building Your Own Compiler." The exercises walk the student through the development of an infix-to-postfix-conversion program and a postfix-expression-evaluation program. We then modify the postfix evaluation algorithm to generate machine-language code. The compiler places this code in a file (using the techniques of Chapter 11). Students can run the machine language produced by their compilers on the software simulators they built in the exercises of Chapter 7!

Chapter 13—The C Preprocessor—provides detailed discussions of the preprocessor directives. The chapter includes detailed information on the `#include` directive (that causes a copy of a specified file to be included in place of the directive in the source code file before the file is compiled) and the `#define` directive that creates symbolic constants and macros. The chapter explains conditional compilation for enabling the programmer to control the execution of preprocessor directives and the compilation of program code. The **#** operator that converts its operand to a string and the **##** operator that concatenates two tokens are discussed. Predefined symbolic constants `__LINE__`, `__FILE__`, `__DATE__` and `__TIME__` are presented. Finally, macro `assert` of the `assert.h` header is discussed. Macro `assert` is valuable in program testing, debugging, verification and validation.

Chapter 14—Other C Topics—presents additional topics including several advanced topics not ordinarily covered in introductory courses. We show how to redirect program input to come from a file, redirect program output to be placed in a file, redirect the output of one program to be the input of another (called "piping"), append the output of a program to an existing file, develop functions that use variable-length argument lists, pass command-line arguments to function `main` and use them in a program, compile programs whose components are spread across multiple files, register functions with `atexit` to be executed at program termination, terminate program execution with function `exit`, use the `const` and `volatile` type qualifiers, specify the type of a numeric constant using the integer and floating-point suffixes, use the signal-handling library to trap unexpected events, create and use dynamic arrays with `calloc` and `realloc`, and use `union`s as a space-saving technique.

Part 2: Object-Based, Object-Oriented and Generic Programming in C++

Chapter 15—C++ as a "Better C"—introduces the non-object-oriented features of C++. These features improve the process of writing procedural programs. The chapter discusses single-line comments, stream input/output, declarations, creating new data types, function prototypes and type checking, `inline` functions (as a replacement for macros), reference parameters, the `const` qualifier, dynamic memory allocation, default arguments, the unary scope resolution operator, function overloading, linkage specifications and function templates.

Chapter 16—C++ Classes and Data Abstraction—begins our discussion of object-based programming. The chapter represents a wonderful opportunity for teaching data abstraction the "right way"—through a language (C++) expressly devoted to implementing abstract data types (ADTs). In recent years, data abstraction has become a major topic in introductory computing courses. Chapters 16 through 18 include a solid treatment of data

abstraction. Chapter 16 discusses implementing ADTs as C++-style `class`es and why this approach is superior to using `struct`s, accessing `class` members, separating interface from implementation, using access functions and utility functions, initializing objects with constructors, destroying objects with destructors, assignment by default memberwise copy and software reusability. One of the chapter exercises challenges the reader to develop a class for complex numbers.

Chapter 17—C++ Classes Part II—continues the study of classes and data abstraction. The chapter discusses declaring and using constant objects, constant member functions, composition—the process of building classes that have objects of other classes as members, `friend` functions and `friend` classes that have special access rights to the `private` and `protected` members of classes, the `this` pointer that enables an object to know its own address, dynamic memory allocation, `static` class members for containing and manipulating class-wide data, examples of popular abstract data types (arrays, strings and queues), container classes and iterators. The chapter exercises ask the student to develop a savings account class and a class for holding sets of integers. We discuss dynamic memory allocation with `new` and `delete`. When `new` fails, it returns a 0 pointer in pre-standard C++. We use this pre-standard style in Chapters 17 through 22. We defer to Chapter 23 the discussion of the new style of `new` failure in which `new` now "throws an exception." We motivate the discussion of `static` class members with a video-game-based example. We emphasize throughout the book and in our professional seminars how important it is to hide implementation details from clients of a class.

Chapter 18—C++ Operator Overloading—is one of the most popular topics in our C++ courses. Students really enjoy this material. They find it a perfect match with the discussion of abstract data types in Chapters 16 and 17. Operator overloading enables the programmer to tell the compiler how to use existing operators with objects of new class types. C++ already knows how to use these operators with objects of built-in types such as integers, floating point numbers and characters. But suppose we create a new string class—what would the plus sign mean when used between string objects? Many programmers use plus with strings to mean concatenation. The chapter discusses the fundamentals of operator overloading, restrictions in operator overloading, overloading with class member functions vs. with nonmember functions, overloading unary and binary operators and converting between types. A feature of the chapter is the substantial case study of an array class, a huge-integer class and a complex numbers class (the last two appear with full source code in the exercises). This material is different from what you do in most programming languages and courses. Operator overloading is a complex topic, but an enriching one. Using operator overloading wisely helps you add that extra "polish" to your classes. With the techniques of Chapters 16, 17 and 18, it is possible to craft a `Date` class that, if we had been using it for the last two decades, could easily have eliminated a major portion of the so-called "Year 2000 (or Y2K) Problem." One of the exercises encourages the reader to add operator overloading to class `Complex` to enable convenient manipulation of objects of this class with operator symbols—as in mathematics—rather than with function calls as the student did in the Chapter 17 exercises.

Chapter 19—C++ Inheritance—deals with one of the most fundamental capabilities of object-oriented programming languages. Inheritance is a form of software reusability in which new classes are developed quickly and easily by absorbing the capabilities of existing classes and adding appropriate new capabilities. The chapter discusses the notions

of base classes and derived classes, `protected` members, `public` inheritance, `protected` inheritance, `private` inheritance, direct base classes, indirect base classes, constructors and destructors in base classes and derived classes and software engineering with inheritance. The chapter compares inheritance (*is a* relationships) with composition (*has a* relationships) and introduces *uses a* and *knows a* relationships. A feature of the chapter is its several substantial case studies. In particular, a lengthy case study implements a point, circle, cylinder class hierarchy. The exercises ask the student to compare the creation of new classes by inheritance vs. composition; to extend the various inheritance hierarchies discussed in the chapter; to write an inheritance hierarchy for quadrilaterals, trapezoids, parallelograms, rectangles and squares; and to create a more general shape hierarchy with two-dimensional shapes and three-dimensional shapes.

Chapter 20—C++ Virtual Functions and Polymorphism—deals with another of the fundamental capabilities of object-oriented programming, namely polymorphic behavior. When many classes are related through inheritance to a common base class, each derived-class object may be treated as a base-class object. This enables programs to be written in a general manner independent of the specific types of the derived-class objects. New kinds of objects can be handled by the same program, thus making systems more extensible. Polymorphism enables programs to eliminate complex `switch` logic in favor of simpler "straight-line" logic. A screen manager of a video game, for example, can simply send a draw message to every object in a linked list of objects to be drawn. Each object knows how to draw itself. A new object can be added to the program without modifying that program as long as that new object also knows how to draw itself. This style of programming is typically used to implement today's popular graphical user interfaces (GUIs). The chapter discusses the mechanics of achieving polymorphic behavior through the use of `virtual` functions. It distinguishes between abstract classes (from which objects cannot be instantiated) and concrete classes (from which objects can be instantiated). Abstract classes are useful for providing an inheritable interface to classes throughout the hierarchy. A feature of the chapter is its polymorphism case study of the point, circle, cylinder shape hierarchy discussed in Chapter 19. The chapter exercises ask the student to discuss a number of conceptual issues and approaches, add abstract classes to the shape hierarchy and develop a basic graphics package—and pursue all these projects with `virtual` functions and polymorphic programming. Our professional audiences insisted that we explain precisely how polymorphism is implemented in C++, and what execution time and memory "costs" one must pay when programming with this powerful capability. We responded by developing an illustration in the section entitled "Polymorphism, `virtual` Functions and Dynamic Binding "Under the Hood" that shows the *vtables* (`virtual` function tables) that the C++ compiler automatically builds to support the polymorphic programming style. We drew these tables in our classes in which we discussed the point, circle, cylinder shape hierarchy. Our audiences indicated that this indeed gave them the information to decide whether polymorphism was an appropriate programming style for each new project they would tackle. We have included this presentation in Section 20.9 and the *vtable* illustration in Fig. 20.2. Please study this presentation carefully. It will give you a much deeper understanding of what is really occurring in the computer when you program with inheritance and polymorphism.

Chapter 21—C++ Stream Input/Output—contains a comprehensive treatment of C++ object-oriented input/output. The chapter discusses the various I/O capabilities of C++ including output with the stream insertion operator, input with the stream extraction oper-

ator, type-safe I/O (a nice improvement over C), formatted I/O, unformatted I/O (for performance), stream manipulators for controlling the stream base (decimal, octal, or hexadecimal), floating-point numbers, controlling field widths, user-defined manipulators, stream format states, stream error states, I/O of objects of user-defined types and tying output streams to input streams (to ensure that prompts actually appear before the user is expected to enter responses). The extensive exercise set asks the student to write various programs that test most of the I/O capabilities discussed in the text.

Chapter 22—C++ Templates—discusses one of the more recent additions to C++. Function templates were introduced in Chapter 15. Class templates enable the programmer to capture the essence of an abstract data type (such as a stack, an array, or a queue) and then create—with minimal additional code—versions of that ADT for particular types (such as a queue of `int`, a queue of `float`, a queue of strings, etc.). For this reason, template classes are often called parameterized types. The chapter discusses using type parameters and nontype parameters and considers the interaction between templates and other C++ concepts, such as inheritance, `friends` and `static` members. The exercises challenge the student to write a variety of function templates and class templates, and to employ these in complete programs.

Chapter 23—C++ Exception Handling—discusses one of the more recent enhancements to the C++ language. Exception handling enables the programmer to write programs that are more robust, more fault tolerant and more appropriate for business-critical and mission-critical environments. The chapter discusses when exception handling is appropriate; introduces the basics of exception handling with `try` blocks, `throw` statements and `catch` blocks; indicates how and when to rethrow an exception; explains how to write an exception specification and process unexpected exceptions; and discusses the important ties between exceptions and constructors, destructors and inheritance. We discuss rethrowing an exception and we illustrate both ways `new` can fail when memory is exhausted. Prior to the C++ draft standard `new` failed by returning 0, much as `malloc` fails in C by returning a `NULL` pointer value.We show the new style of `new` failing by throwing a `bad_alloc` (bad allocation) exception. We illustrate how to use `set_new_handler` to specify a custom function to be called to deal with memory exhaustion situations. We discuss the `auto_ptr` class template to guarantee that dynamically allocated memory will be properly `deleted` to avoid memory leaks.

Part 3: Object-Oriented, GUI Event-Driven, Graphics and Multimedia Programming in Java

Chapter 24—Introduction to Java Applications and Applets—introduces a typical Java programming environment and provides a lightweight introduction to programming applications and applets in the Java programming language. Some of the input and output is performed using a new graphical user interface (GUI) element called `JOptionPane` that provides predefined windows (called dialogs) for input and output. `JOptionPane` handles ouputting data to windows and inputting data from windows. The chapter introduces applets using several of the sample demonstration applets supplied with the Java 2 Software Development Kit (J2SDK). We use `appletviewer` (a utility supplied with the J2SDK) to execute several sample applets. We then write Java applets that perform tasks similar to the applications written earlier in the chapter, and we explain the similarities and differences between applets and applications. After studying this chapter, the student will understand how to

write simple, but complete, Java applications and applets. The next several chapters use both applets and applications to demonstrate additional key programming concepts.

Chapter 25—Beyond C & C++: Operators, Methods & Arrays—focuses on both the similarities and differences among Java, C and C++. The chapter discusses the primitive types in Java and how they differ from C/C++, as well as some differences in terminology. For example, what we call a function in C/C++ is called a method in Java. The chapter also contains a discussion of logical operators—&& (logical AND), & (boolean logical AND), || (logical OR), | (boolean logical inclusive OR), ^ (boolean logical exclusive OR) and ! (NOT) applications. The topic of method overloading (as compared to function overloading in C++) is motivated and explained. In this chapter, we also introduce events and event handling—elements required for programming graphical user interfaces. Events are notifications of state change such as button clicks, mouse clicks, pressing a keyboard key, etc. Java allows programmers to respond to various events by coding methods called event handlers. We also introduce arrays in Java, which are processed as full-fledged objects. This is further evidence of Java's commitment to nearly 100% object-orientation. We discuss the structuring of data into arrays, or groups, of related data items of the same type. The chapter presents numerous examples of both single-subscripted arrays and double-subscripted arrays.

Chapter 26—Java Object-Based Programming—begins our deeper discussion of classes. The chapter focuses on the essence and terminology of classes and objects. What is an object? What is a class of objects? What does the inside of an object look like? How are objects created? How are they destroyed? How do objects communicate with one another? Why are classes such a natural mechanism for packaging software as reusable componentry? The chapter discusses implementing abstract data types as Java-style classes, accessing class members, enforcing information hiding with `private` instance variables, separating interface from implementation, using access methods and utility methods, initializing objects with constructors and using overloaded constructors. The chapter discusses declaring and using constant references, composition—the process of building classes that have as members references to objects, the `this` reference that enables an object to "know itself," dynamic memory allocation, `static` class members for containing and manipulating class-wide data and examples of popular abstract data types such as stacks and queues. The chapter also introduces the `package` statement and discusses how to create reusable packages. The chapter exercises challenge the student to develop classes for complex numbers, rational numbers, times, dates, rectangles, huge integers, a class for playing Tic-Tac-Toe, a savings account class and a class for holding sets of integers.

Chapter 27—Java Object-Oriented Programming—discusses the relationships among classes of objects, and programming with related classes. How can we exploit commonality between classes of objects to minimize the amount of work it takes to build large software systems? What is polymorphism? What does it mean to "program in the general" rather than "programming in the specific?" How does programming in the general make it easy to modify systems and add new features with minimal effort? How can we program for a whole category of objects rather than programming individually for each type of object? The chapter deals with one of the most fundamental capabilities of object-oriented programming languages, inheritance, which is a form of software reusability in which new classes are developed quickly and easily by absorbing the capabilities of existing classes and adding appropriate new capabilities. The chapter discusses the notions of superclasses and subclasses, `protected` members, direct superclasses, indirect superclasses, use of

constructors in superclasses and subclasses, and software engineering with inheritance. We introduce inner classes that help hide implementation details. Inner classes are most frequently used to create GUI event handlers. Named inner classes can be declared inside other classes and are useful in defining common event handlers for several GUI components. Anonymous inner classes are declared inside methods and are used to create one object—typically an event handler for a specific GUI component. The chapter compares inheritance (*is a* relationships) with composition (*has a* relationships). A feature of the chapter is its case study implementation of a point, circle, cylinder class hierarchy. The exercises ask the student to compare the creation of new classes by inheritance vs. composition; to extend the inheritance hierarchies discussed in the chapter; to write an inheritance hierarchy for quadrilaterals, trapezoids, parallelograms, rectangles and squares; and to create a more general shape hierarchy with two-dimensional shapes and three-dimensional shapes. The chapter explains polymorphic behavior. When many classes are related through inheritance to a common superclass, each subclass object may be treated as a superclass object. This enables programs to be written in a general manner independent of the specific types of the subclass objects. New kinds of objects can be handled by the same program, thus making systems more extensible. Polymorphism enables programs to eliminate complex `switch` logic in favor of simpler "straight-line" logic. A video game screen manager, for example, can send a "draw" message to every object in a linked list of objects to be drawn. Each object knows how to draw itself. A new type of object can be added to the program without modifying that program as long as that new object also knows how to draw itself. This style of programming is typically used to implement today's popular graphical user interfaces. The chapter distinguishes between `abstract` classes (from which objects cannot be instantiated) and concrete classes (from which objects can be instantiated). The chapter also introduces interfaces—sets of methods that must be defined by any class that `implements` the interface.

Chapter 28—Java Graphics and Java2D—begins a run of three chapters that present the multimedia "sizzle" of Java. Traditional C and C++ programming are pretty much confined to character-mode input/output. Some versions of C++ are supported by platform-dependent class libraries that can do graphics, but using these libraries makes your applications nonportable. Java's graphics capabilities are platform independent and hence, portable—and we mean portable in a worldwide sense. You can develop graphics-intensive Java applets and distribute them over the World Wide Web to colleagues everywhere and they will run nicely on the local Java platforms. We discuss graphics contexts and graphics objects; drawing strings, characters and bytes; color and font control; screen manipulation and paint modes; and drawing lines, rectangles, rounded rectangles, 3-dimensional rectangles, ovals, arcs and polygons. We introduce the Java2D API, new in Java 2, which provides powerful graphical manipulation tools. The chapter has many figures that painstakingly illustrate each of these graphics capabilities with live-code examples, appealing screen outputs, detailed features tables and detailed line art.

Chapter 29—Java Graphical User Interface Components—introduces the creation of applets and applications with user-friendly graphical user interfaces (GUIs). This chapter focuses on Java's new Swing GUI components. These platform-independent GUI components are written entirely in Java. This provides Swing GUI components with great flexibility—they can be customized to look like the computer platform on which the program executes, or they can use the standard Java look-and-feel that provides an identical

user interface across all computer platforms. We discuss the `javax.swing` package, which provides especially powerful GUI components. The chapter illustrates GUI design principles, the `javax.swing` hierarchy, labels, push buttons, text fields, text areas, combo boxes, check boxes, panels, scrolling panels, custom panels, handling mouse events, windows, menus and using three of Java's simpler GUI layout managers: `FlowLayout`, `BorderLayout` and `GridLayout`. The chapter concentrates on Java's delegation event model for GUI processing. The exercises challenge the student to create specific GUIs, exercise various GUI features, develop drawing programs that let the user draw with the mouse and control fonts.

Chapter 30—Java Multimedia: Images, Animation, Audio and Video—deals with Java's capabilities for making computer applications "come alive." It is remarkable that students in first programming courses will be writing applications with all these capabilities. The possibilities are intriguing. Students now access (over the Internet and through CD-ROM technology) vast libraries of graphics images, audios and videos, and can weave their own together with those in the libraries to form creative applications. Already most new computers come "multimedia equipped." Dazzling term papers and classroom presentations are being prepared by students with access to vast public domain libraries of images, drawings, voices, pictures, videos, animations and the like. A "paper" when most of us were in the earlier grades was a collection of characters, possibly handwritten, possibly typewritten. A "paper" can be a multimedia "extravaganza." It can hold your interest, pique your curiosity, make you feel what the subjects of the paper felt when they were making history. Multimedia can make your science labs much more exciting. Textbooks can come alive. Instead of looking at a static picture of some phenomenon, you can watch that phenomenon occur in a colorful, animated, presentation with sounds, videos and various other effects. People can learn more, learn it in more depth and experience more viewpoints. A feature of the chapter is the image maps discussion that enable a program to sense the presence of the mouse pointer over a region of an image, without clicking the mouse. We present a live-code image map application with the icons Prentice Hall artists created for our *Java Multimedia Cyber Classroom* programming tips. As the user moves the mouse pointer across the six icon images, the type of tip is displayed, either "Good Programming Practice" for the thumbs-up icon, "Portability Tip" for the bug with the suitcase icon, and so on.

Part 4: Appendices

Several Appendices provide valuable reference material. We present Internet and Web resources for C, C++ and Java in Appendix A; a list of C99 Internet and Web resources in Appendix B; complete operator precedence and associativity charts for C, C++ and Java in Appendix C; the set of ASCII character codes in Appendix D. Appendix E is a complete tutorial on number systems including many self-review exercises with answers. Appendix F provides an overview of the C Standard Libraries and Web resources for these libraries.

Acknowledgments

One of the great pleasures of writing a textbook is acknowledging the efforts of many people whose names may not appear on the cover, but whose hard work, cooperation, friendship and understanding were crucial to the production of the book. Many people at Deitel & Associates, Inc. devoted long hours to this project.

- Abbey Deitel, President
- Barbara Deitel, Chief Financial Officer
- Christi Kelsey, Director of Business Development
- Jeff Listfield, Senior Developer
- Su Zhang, Senior Developer

We would also like to thank the participants in the Deitel & Associates, Inc., College Internship Program: Mike Oliver, Brian O'Connor and Adam Burke who worked on the book's ancillary package.[1] We would especially like to thank Tim Christensen. Tim, a senior at Boston College majoring in Business Management with a concentration in Computer Science, tested all the source code for the entire book, added comments to all of the C code (Chapters 2–14) and updated the programs to our standard coding conventions. He created Appendix F (C Standard Libraries Internet and Web Resources) and revised Appendix B (C99 Internet and Web Resources). Tim also worked on the ancillary package for the textbook.

We are fortunate to have worked on this project with the talented and dedicated team of publishing professionals at Prentice Hall. We especially appreciate the extraordinary efforts of our Computer Science Editor, Kate Hargett and her boss and our mentor in publishing—Marcia Horton, Editorial Director of Prentice-Hall's Engineering and Computer Science Division. Vince O'Brien and Tom Manshreck did a marvelous job managing the production of the book. Sarah Parker managed the publication of the book's extensive ancillary package.

We wish to acknowledge the efforts of our *Fourth Edition* reviewers and to give a special note of thanks to Carole Snyder of Prentice Hall who managed this extraordinary review effort. [Please note that the first two editions of *C How to Program* included only C and C++; Java was added in the *Third Edition*.]

- Rex Jaeschke (Independent Consultant; former chair of the ANSI C Committee)
- John Benito (Convener of the ISO working group that is responsible for the C programming language)
- Deena Engel (New York University)
- Geb Thomas (University of Iowa)
- Jim Brzowski (University of Massachusetts – Lowell)

We wish to acknowledge again the efforts of our previous edition reviewers (some first edition, some second edition, some third edition and some all three); the affiliations were current at the time of the review):

---

1. This highly competitive program (we received 1000+ applications for 11 internship positions in 2003) offers a limited number of salaried positions to Boston-area college students majoring in Computer Science, Information Technology, Marketing, Management and English. Students work at our corporate headquarters in Maynard, Massachusetts full-time in the summers and (for those attending college in the Boston area) part-time during the academic year. We also offer full-time internship positions for students interested in taking a semester off from school to gain industry experience. Regular full-time positions are available from time to time to college graduates. For more information, please contact our president—`abbey.deitel@deitel.com`—and visit our Web site, `www.deitel.com`.

- Rex Jaeshke (Independent Consultant; former chair of the ANSI C Committee)
- Randy Meyers (NetCom; ANSI C Committee Chair; former ANSI C++ Committee Member)
- Simon North (Synopsis, XML Author)
- Fred Tydeman (Consultant)
- Kevin Wayne (Princeton University)
- Eugene Katzin (Montgomery College)
- Sam Harbison (Texas Instruments, PH Author)
- Chuck Allison (Tydeman Consulting)
- Catherine Dwyer (Pace University)
- Glen Lancaster (DePaul University)
- David Falconer (California State University at Fullerton)
- David Finkel (Worcester Polytechnic)
- H. E. Dunsmore (Purdue University)
- Jim Schmolze (Tufts University)
- Gene Spafford (Purdue University)
- Clovis Tondo (IBM Corporation and visiting professor at Nova University)
- Jeffrey Esakov (University of Pennsylvania)
- Tom Slezak (University of California, Lawrence Livermore National Laboratory)
- Gary A. Wilson (Gary A Wilson & Associates and University of California Berkeley Extension)
- Mike Kogan (IBM Corporation; chief architect of 32-bit OS/2 2.0)
- Don Kostuch (IBM Corporation retired; worldwide instructor in C, C++ and object-oriented programming)
- Ed Lieblein (Nova University)
- John Carroll (San Diego State University)
- Alan Filipski (Arizona State University)
- Greg Hidley (University of California, San Diego)
- Daniel Hirschberg (University of California, Irvine)
- Jack Tan (University of Houston)
- Richard Alpert (Boston University)
- Eric Bloom (Bentley College)

These reviewers scrutinized every aspect of the text and made countless suggestions for improving the accuracy and completeness of the presentation.

Contacting Deitel & Associates

We would sincerely appreciate your comments, criticisms, corrections and suggestions for improving the text. Please address all correspondence to:

```
deitel@deitel.com
```

We will respond promptly.

Errata
We will post all errata for the *Fourth Edition* at `www.deitel.com`.

Customer Support
Please direct all software and installation questions to Pearson Education Technical Support:

- By phone: 1-800-677-6337
- By email: `media.support@pearsoned.com`
- On the Web: `247.prenhall.com`

Please direct all C, C++ and Java language questions to `deitel@deitel.com`. We will respond pomptly.

Welcome to the exciting worlds of procedural programming in C; object-based, object-oriented and generic programming in C++; and graphics, graphical user interface, multimedia and event-driven programming in Java. We sincerely hope you enjoy learning with this book.

Dr. Harvey M. Deitel
Paul J. Deitel

About the Authors

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 42 years experience in the computing field, including extensive industry and academic experience. Dr. Deitel earned B.S. and M.S. degrees from the Massachusetts Institute of Technology and a Ph.D. from Boston University. He worked on the pioneering virtual-memory operating-systems projects at IBM and MIT that developed techniques now widely implemented in systems such as UNIX, Linux and Windows XP. He has 20 years of college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., with his son, Paul J. Deitel. He and Paul are the co-authors of several dozen books and multimedia packages and they are writing many more. With translations published in Japanese, German, Russian, Spanish, Traditional Chinese, Simplified Chinese, Korean, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish, the Deitels' texts have earned international recognition. Dr. Deitel has delivered professional seminars to major corporations, government organizations and the military.

Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of the Massachusetts Institute of Technology's Sloan School of Management, where he studied Information Technology. Through Deitel & Associates, Inc., he has delivered C, C++, Java, Internet and World Wide Web courses to industry clients, including IBM, Sun Microsystems, Dell, Lucent Technologies, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, Compaq, White Sands Missile Range, Rogue Wave Software, Boeing, Stratus, Cambridge Technology Partners, Open Environment Corporation, One Wave, Hyperion Software, Adra Systems, Entergy, Cable-Data Systems and many other organizations. He has lectured on C++ and Java for the Boston Chapter of the Association for Computing Machinery and has taught satellite-based Java courses through a cooperative venture of Deitel & Associates, Prentice Hall and the

Technology Education Network. He and his father, Dr. Harvey M. Deitel, are the world's best-selling Computer Science textbook authors.

About Deitel & Associates, Inc.

Deitel & Associates, Inc., is an internationally recognized corporate training and content-creation organization specializing in Internet/World Wide Web software technology, e-business/e-commerce software technology, object technology and computer programming languages education. The company provides instructor-led courses on Internet and World Wide Web programming, wireless Internet programming, object technology, and major programming languages and platforms, such as C, C++, Visual C++® .NET, Visual Basic® .NET, C#, Java, Advanced Java, XML, Perl, Python and more. The founders of Deitel & Associates, Inc., are Dr. Harvey M. Deitel and Paul J. Deitel. The company's clients include many of the world's largest computer companies, government agencies, branches of the military and business organizations. Through its 27-year publishing partnership with Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks, professional books, interactive CD-based multimedia *Cyber Classrooms*, *Complete Training Courses*, Web-based training courses and course management systems e-content for popular CMSs such as WebCT™, Blackboard™ and CourseCompass^{SM}. Deitel & Associates, Inc., and the authors can be reached via e-mail at:

```
deitel@deitel.com
```

To learn more about Deitel & Associates, Inc., its publications and its worldwide corporate on-site training curriculum, see the last few pages of this book or visit:

```
www.deitel.com
```

Individuals wishing to purchase Deitel® books, *Cyber Classrooms*, *Complete Training Courses* and Web-based training courses can do so through bookstores, online booksellers and:

```
www.deitel.com
www.prenhall.com/deitel
www.InformIT.com/deitel
www.InformIT.com/cyberclassrooms
```

Bulk orders by corporations and academic institutions should be placed directly with Prentice Hall. See the last few pages of this book for worldwide ordering instructions.