国外计算机科学教材系列

# C++ How to Program, Fourth Edition

PEARSON Prentice Hall

CD includes Microsoft® Visual C++® 6.0 Introductory Edition

# C++大学教程

## （第四版）（英文版）

Introducing

## Web Programming with CGI and Object-Oriented Design with the UML™

- ANSI/ISO Standard C++
- Control Structures
- Functions/Arrays
- Pointers/Strings
- Vector/String Objects
- Classes/Data Abstraction
- Operator Overloading
- Inheritance/Polymorphism
- Virtual Functions
- Run-Time Type Information
- Standard Stream I/O
- Templates/STL
- Exception Handling
- Files/Data Structures
- Bits/Characters
- Namespaces
- Internet/Web Resources

**UML OOD CASE STUDY**

- Use Case Diagrams
- Class Diagrams
- Object Diagrams
- Association
- Composition
- Statechart Diagrams
- Activity Diagrams
- Sequence Diagrams
- Collaboration Diagrams


DEITEL

[美] Deitel Deitel 著

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

国外计算机科学教材系列

# C++ 大学教程

## （第四版）（英文版）

# C++ How to Program

## Fourth Edition


H. M Deitel

Deitel & Associates, Inc.

[美] 著

P. J. Deitel

Deitel & Associates, Inc.



電子工業出版社

**Publishing House of Electronics Industry**

北京 · BEIJING

## 内容简介

本书的作者Deitel一家是美国编程教材方面的名家，他们的作品繁多，并且多为美国各著名大学的指定教材。本书是一本C++编程方面的优秀教程，全面介绍了过程式编程与面向对象编程的原理与方法，细致地分析了各种性能问题、可移植性问题以及可能出错的情况。作者通过大量的示例程序，重点突出了利用UML进行面向对象的设计，引入了使用CGI的Web应用开发，并且帮助学生创建真实世界的C++应用程序。本书无论从广度和深度上来说都非常全面，并且从基础概念讲起，同样适合没有编程经验的读者学习。通过可以实际使用的完整例子，可以使读者潜移默化地掌握概念。

本书可作为高等院校相关专业的编程语言教材和C++编程教材，也是软件设计人员进行C++程序开发的宝贵参考资料。



版权贸易合同登记号　图字：01-2005-3589

**图书在版编目（CIP）数据**

C++大学教程 = C++ How to Program：第4版 /（美）戴特尔（Deitel, H. M.）等著.
北京：电子工业出版社，2005.9
（国外计算机教学教材系列）
ISBN 7-121-01759-8

Ⅰ. C… Ⅱ. 戴… Ⅲ. C语言－程序设计－高等学校－教材－英文 Ⅳ. TP312

中国版本图书馆CIP数据核字（2005）第106211号

责任编辑：冯小贝
印　　刷：北京天竺颖华印刷厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
经　　销：各地新华书店
开　　本：787 × 980　1/16　　印张：87.75　　字数：1966千字
印　　次：2005年9月第1次印刷
定　　价：128.00元（附光盘1张）

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换；若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

# Preface

Welcome to ANSI/ISO Standard C++! 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 C++ educational experience. As the book goes to publication, it is compliant with the latest version of the ANSI/ISO C++ Standard (one of the most important worldwide standards for the computing community) and with object-oriented design using the latest version of the UML (Unified Modeling Language) from the Object Management Group (OMG). We tuned the writing, the pedagogy, our coding style, the book's ancillary package and even added a substantial treatment of developing Internet- and Web-based applications. We have added a comprehensive Tour of the Book section to Chapter 1. This will help instructors, students and professionals get a sense of the rich coverage the book provides of C++ object-oriented programming, object-oriented design with the UML and generic programming. If you are evaluating the book, please read the Tour of the Book now in pages 44–56.

Whether you are an instructor, a student, an experienced professional or a novice programmer, this book has much to offer. C++ is a world-class programming language for developing industrial-strength, high-performance computer applications. We carefully audited the manuscript against the ANSI/ISO C++ standard document,[1] which defines C++, and we were privileged to have as a reviewer Steve Clamage of Sun Microsystems who heads the ANSI J16 Committee responsible for evolving the C++ standard. As a result, the programs you create by studying this text should port easily to any ANSI/ISO-compliant compiler.

1. An electronic PDF copy of the C++ standard document, number ISO-IEC 14882-1998, is available for $18 at **webstore.ansi.org/ansidocstore/default.asp**; a paper copy is available from this site for $175.

In this Preface, we overview *C++ How to Program, 4/e*'s comprehensive suite of educational materials that help instructors maximize their students' C++ learning experience. We explain conventions we use, such as syntax coloring the code examples, "code washing" and highlighting important code segments to help focus students' attention on the key concepts introduced in each chapter. We overview the new features of *C++ How to Program, 4/e*, including our early treatment of arrays and strings as objects, an enhanced treatment of object-oriented programming, Web-application development with CGI, the enhanced elevator-simulation object-oriented design (OOD) case study with the UML, and the extensive use of UML diagrams that have been upgraded to UML version 1.4 standards.

Prentice Hall has bundled *Microsoft's Visual C++® 6 Introductory Edition* software with the text and offers a separate value-pack containing *C++ How to Program, 4/e*, with Metrowerks *CodeWarrior* for the Macintosh and Windows. We list several compilers that are available on the Web free for download. To further support novice programmers, we offer six 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++ programs using various popular C++ development environments.

We overview the complete package of ancillary materials available to instructors and students using *C++ How to Program, 4/e*. 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 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. Numerous support materials are available for students at the Companion Web Site, as well. For instructors who want to hold closed-lab sessions (or highly structured homework assignments), we provide the optional, for-sale manual, *C++ in the Lab*. This publication includes carefully constructed Prelab Activities, Lab Exercises and Postlab Activities.

This Preface also discusses *The C++ Multimedia Cyber Classroom, 4/e*, an interactive, multimedia CD-based version of the book. This learning aid provides audio "walk-throughs" of programs, animations of programs executing and hundreds of exercises and solutions. We describe how to order both the Cyber Classroom and *The Complete C++ Training Course, 4/e*, boxed product, which contains the Cyber Classroom and the textbook.

We discuss several DEITEL e-learning initiatives, including an explanation of Deitel content available for the *Blackboard*, *CourseCompass* and *WebCT* Course Management Systems, each of which supports *C++ How to Program, 4/e*. *Premium CourseCompass*, which offers enhanced Deitel content based on *The C++ Multimedia Cyber Classroom, 4/e*, will be available in January 2003.

*C++ How to Program, 4/e*, was reviewed by 52 distinguished academics and industry professionals; 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**. 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

This book contains many features including:

***Full-Color Presentation***
This book is in full color to show programs and their outputs as they typically would appear on a computer screen. We syntax color all the C++ code, as do many C++ integrated-development environments and code editors. This greatly improves code readability—an especially important goal, given that this book contains over 20,000 lines of code. Our syntax-coloring conventions are as follows:

```
comments appear in green
keywords appear in dark blue
errors appear in red
constants and literal values appear in light blue
all other code appears in black
```

***Code Highlighting and User-Input Highlighting***
We have added extensive code highlighting. In our code walkthroughs (at Deitel, we call these "writearounds"), we have eliminated most of the "redundant" code snippets that appeared inline in the text in the *Third Edition*. We kept them in the earliest portion of the book as a pedagogic device to help novices. We want the reader to see all new code features in context, so from Chapter 3 forward, our code walkthroughs simply refer to the line numbers of the new code segments inside complete source programs. To make it easier for readers to spot the featured segments, we have highlighted them in bright yellow. 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 done extensive "code washing" of all the source code programs in the text, the lab manual, the ancillaries and the *Cyber Classroom*.

***Early Introduction of Standard Library* `string` *and* `vector` *Objects***
Object-oriented programming languages generally offer the ability to create string and array objects by instantiating them from library classes or from programmer-defined classes. It is also important for students learning C++ to become familiar with C-style, pointer-based arrays and strings, because of the massive amount of C and early C++ legacy code they will encounter in industry. In *C++ How to Program, 4/e*, we show all three means of creating strings and arrays. In Chapters 4 and 5 we show the traditional, C-like pointer-based arrays and strings, respectively. In Chapter 8, Operator Overloading, we create our own user-defined classes **`Array`** and **`String`**. At the end of Chapter 8, we introduce library classes **`vector`** and **`string`**, which we explain in detail in Chapter 15 and Chapter 21, respectively. Through Chapter 8, we favor pointer-based arrays and strings; after Chapter 8, we favor the library classes. The Chapter 15 material on **`string`** could be taught at any point after Chapter 8. The Chapter 21 material on **`vector`** (and other aspects of the STL) could also reasonably be taught after Chapter 8, although we recommend covering Chapter 11, Templates, first.

***Tuned Treatment of Object-Oriented Programming in Chapters 9 and 10***

This is one of the most significant improvements in this new edition. We performed a high-precision upgrade to Chapters 9 and 10. The improvements make the material clearer and more accessible to students and professionals, especially those studying object-orientation for the first time.

***Redesigned Pedagogy of Chapter 9, Object-Oriented Programming: Inheritance.*** The new Chapter 9 carefully walks the reader through a five-example sequence that demonstrates **`private`** data, **`protected`** data and software reuse via inheritance. We begin by demonstrating a class with **`private`** data members and **`public`** member functions to manipulate that data. Next, we implement a second class with several additional capabilities. To do this, we duplicate much of the first example's code. In our third example, we begin our discussion of inheritance and software reuse—we use the class from the first example as a base class and inherit its data and functionality into a new derived class. This example introduces the inheritance mechanism and demonstrates that a derived class cannot access its base class's **`private`** data directly. This motivates our fourth example, in which we introduce **`protected`** data in the base class and demonstrate that the derived class can indeed access its base class's **`protected`** data. The last example in the sequence demonstrates proper software engineering by defining the base class's data as **`private`** and using the base class's **`public`** member functions (that were inherited by the derived class) to manipulate the base class's **`private`** data from the derived class. We follow the five-part introduction with a three-level class hierarchy that employs the software engineering techniques introduced earlier in the chapter. The chapter closes with a discussion of the three inheritance types supported by C++ and a general discussion of software engineering with inheritance.

***Redesigned Pedagogy of Chapter 10, Object-Oriented Programming: Polymorphism.*** The new Chapter 10 builds on the inheritance concepts presented in Chapter 9 and focuses on the relationships between classes in a class hierarchy. Chapter 10 uses a four-example sequence to present the powerful processing capabilities that these relationships enable. We begin with an example that illustrates the "is-a" relationship between a derived-class object and its base-class type. This relationship enables the derived-class object to be treated as an object of its base class. We show that we are able to aim a base-class pointer at a derived-class object and invoke the base-class's functions on that object. In our second example, we demonstrate that the reverse is not true—a base-class object is not considered to be an object of its derived-class type—and we show that compiler errors occur if a program attempts to manipulate a base-class object in this manner. Our third example demonstrates that the only functions which can be invoked through a base-class pointer are those functions defined by the base class. The example shows that attempts to invoke derived-class-only functions result in error messages. The last example in the sequence introduces polymorphism with virtual functions, which enable a program to process objects of classes related by a class hierarchy as objects of their base-class type. When a virtual function is invoked via a base-class pointer (or reference), the derived-class-specific version of that function is invoked. The chapter continues with a case study on polymorphism in which we process an array of objects that all have a common abstract base class that contains the set of functions common to every class in the hierarchy. We follow this example with an in-depth discussion of how polymorphism works "under the hood." We conclude with a case study that demonstrates how a program that processes objects polymorphically can still

perform type-specific processing by determining at execution time the type of the object currently being processed.

***Web Applications Development with CGI***

The new Chapter 16, Web Programming with CGI, has everything readers need to begin developing their own Web-based applications that will run on the Internet![2] Readers will learn how to build so-called *n*-tier applications, in which the functionality provided by each tier can be distributed to separate computers across the Internet or executed on the same computer. In particular, we build a three-tier online bookstore application. The bookstore's information is stored in the application's data tier. In industrial-strength applications, the data tier is typically a database such as Oracle, Microsoft® SQL Server or MySQL. For simplicity, we use text files and employ the file-processing techniques of Chapter 14 to access these files. The user enters requests and receives responses at the application's client tier, which is typically a computer running a Web browser such as Microsoft Internet Explorer or Netscape®. Web browsers, of course, know how to communicate with Web sites throughout the Internet. The middle tier contains both a Web server and an application-specific C++ program (e.g., our bookstore application). The Web server communicates with the C++ program (and vice versa) via the CGI (Common Gateway Interface) protocol. We use the popular Apache HTTP server as our Web server, which is available free for download from **www.apache.org**. The Web server knows how to communicate with the client tier across the Internet using the HyperText Transfer Protocol (HTTP). We discuss the crucial role of the Web server in Web programming and provide a simple example that requests a Web page from a Web server. We discuss CGI and how it allows a Web server to communicate with the top tier and CGI scripts (i.e., our C++ programs). We provide a simple example that gets the time and date from the server and renders it in a browser. In our forms-based examples we use buttons, password fields, check boxes and text fields. We present an example of an interactive portal for a travel company that displays airfares to various cities. Travel-club members can log in and view discounted airfares. We also discuss various methods of storing client-specific data, which include hidden fields (i.e., information stored in a Web page but not rendered by the Web browser) and cookies—small text files that the browser stores on the client's machine. The chapter examples conclude with an e-business case study of an online bookstore that allows users to add books to an electronic shopping cart. This case study contains several CGI scripts that interact with one another to form a complete application. The online bookstore is password protected, so users first must log in to gain access.

***XHTML***

The World Wide Web Consortium (W3C) has declared HyperText Markup Language (HTML) to be a legacy technology that will undergo no further development. HTML is being replaced by the Extensible HyperText Markup Language (XHTML)—an XML-based technology that rapidly is becoming the standard for describing Web content. We use XHTML in Chapter 16, Web Programming with CGI; Appendix E presents an XHTML introduction. If you are not familiar with XHTML, please read Appendix E before reading Chapter 16.

---

2. There are other technologies for developing Web-based applications. Java developers use Java servlets and JavaServer Pages. Windows-platform developers use Active Server Pages (ASP). We chose CGI for this book, because both standard C++ and CGI are platform independent.

***Unified Modeling Language (UML)***

The Unified Modeling Language (UML) has become the preferred graphical modeling language for designing object-oriented systems. In *C++ How to Program, Third Edition*, we used the UML in optional sections only, and we used conventional flowchart segments and inheritance diagrams to reinforce the explanations. We have fully converted the diagrams in the book to be UML 1.4 compliant. In particular, we upgraded all the figures in the UML/OOD Elevator Simulation case study; we converted all the flowcharts in Chapter 2, Control Structures, to UML activity diagrams; and we converted all the inheritance diagrams in Chapters 9, 12, 14 and 22 to UML class diagrams.

This *Fourth Edition* carefully tunes the optional (but highly recommended) case study we present on object-oriented design using the UML. In the case study, we fully implement an elevator simulation. In the "Thinking About Objects" sections at the ends of Chapters 1–7 and 9, we present a carefully paced introduction to object-oriented design using the UML. We present a concise, simplified subset of the UML then guide the reader through a first design experience intended for the novice object-oriented designer/programmer. The case study is fully solved. It is not an exercise; rather, it is an end-to-end learning experience that concludes with a detailed walkthrough of the C++ code. In each of the first five chapters, we concentrate on the "conventional" methodology of structured programming, because the objects that we build will use these structured-program pieces. We conclude each chapter with a "Thinking About Objects" section, in which we present an introduction to object orientation using the UML. These "Thinking About Objects" sections help students develop an object-oriented way of thinking, so that they immediately can use the object-oriented programming concepts they begin learning in Chapter 6. In the first of these sections at the end of Chapter 1, we introduce basic concepts (i.e., "object think") and terminology (i.e., "object speak"). In the optional "Thinking About Objects" sections at the ends of Chapters 2–5, we consider more substantial issues, as we undertake a challenging problem with the techniques of object-oriented design (OOD). We analyze a typical problem statement that requires a system to be built, determine the objects needed to implement that system, determine the attributes these objects need to have, determine the behaviors these objects need to exhibit and specify how the objects need to interact with one another to meet the system requirements. We accomplish this even before we discuss how to write object-oriented C++ programs. In the "Thinking About Objects" sections at the ends of Chapters 6, 7 and 9, we build a C++ implementation of the object-oriented system we designed in the earlier chapters. This project enabled us to incorporate topics that we do not discuss in any other section of the book, including object interaction, an in-depth discussion of handles, the philosophy of using references vs. pointers and the use of forward declarations to avoid circular-include problems. This case study will help prepare students for the kinds of substantial projects they will encounter in industry. We employ a carefully developed, incremental object-oriented design process to produce a UML-based design for our elevator simulator. From this design, we produce a substantial working C++ implementation using key programming notions, including classes, objects, encapsulation, visibility, composition and inheritance.

***More About the (Optional) Elevator Simulation Case Study***

This case study was introduced in *C++ How to Program, 3/e*, and was carefully tuned for the *Fourth Edition*. We brought all the UML diagrams into compliance with version 1.4, we reorganized many of the diagrams to make them clearer, we code washed the complete

C++ solution presented in the book, and we tuned the discussions for clarity and precision. The case study was submitted to a distinguished team of OOD/UML reviewers, including leaders in the field from Rational (the creators of the UML) and the Object Management Group (responsible for maintaining and evolving the UML).

In Chapter 2, we begin the first phase of the object-oriented design (OOD) for our elevator simulator—identifying the classes needed to implement the simulator. We also introduce the UML use case, class and object diagrams and the concepts of associations, multiplicity, composition, roles and links. In Chapter 3, we determine many of the class attributes needed to implement the elevator simulator. We also introduce the UML statechart and activity diagrams and the concepts of events and actions as they relate to these diagrams. In Chapter 4, we determine many of the operations (behaviors) of the classes in the elevator simulation. We also introduce the UML sequence diagram and the concept of messages sent between objects. In Chapter 5, we determine the collaboration (sets of interactions among objects in the system) needed to implement the elevator system and represent these interactions using the UML collaboration diagram. We also include a bibliography and a list of Internet and Web resources that contain the UML 1.4 specifications and other reference materials, general resources, tutorials, FAQs, articles, whitepapers and software. In Chapter 6, we use the UML class diagram developed in previous sections to outline the C++ header files that define our classes. We also introduce the concept of handles to objects in the system, and we begin to study how to implement handles in C++. In Chapter 7, we present a complete elevator simulator C++ program (approximately 1200 lines of code) and a detailed code walkthrough. The code follows directly from the UML-based design created in previous sections and employs our best programming practices. We also discuss dynamic-memory allocation, composition, object interaction via handles, and how to use forward declarations to avoid the circular-include problem. In Chapter 9, we update the elevator simulation design and implementation to incorporate inheritance and suggest further modifications.

***Standard Template Library (STL)***

This might be one of the most important chapters in the book in terms of your appreciation of software reuse. The STL defines powerful, template-based, reusable components that implement many common data structures and algorithms used to process those data structures. Chapter 21 introduces the STL and discusses its three key components—containers, iterators and algorithms. STL containers are data structures capable of storing objects of any data type. We show that there are three container categories—first-class containers, adapters and near containers. STL iterators, which are similar to pointers (but much safer), are used by programs to manipulate the STL-container elements. In fact, standard arrays can be manipulated as STL containers, using standard pointers as iterators. We show that manipulating containers with iterators is convenient and provides tremendous expressive power when combined with STL algorithms—in some cases, reducing many lines of code to a single statement. STL algorithms are functions that perform common data manipulations such as searching, sorting, comparing elements (or entire data structures), etc. There are approximately 70 algorithms implemented in the STL; these include common container operations such as searching for an element, sorting elements, comparing elements, removing elements, replacing elements and many more. Most of these algorithms use iterators to access container elements. We show that each first-class container supports specific iterator types, some of which are more powerful than others. A container's supported iterator type

determines whether the container can be used with a specific algorithm. Iterators encapsulate the mechanism used to access container elements. This encapsulation enables many of the STL algorithms to be applied to a variety of containers without regard for the underlying container implementation. As long as a container's iterators support the minimum requirements of the algorithm, the algorithm can process that container's elements. This also enables programmers to create algorithms that can process the elements of multiple container types. An advantage of the STL is that programmers can reuse the STL containers, iterators and algorithms to implement common data representations and manipulations. This reuse saves substantial development time and resources.

## Teaching Approach

Our book is intended to be used at the introductory and intermediate levels. We have not attempted to cover every feature of the C++ standard. C++ has replaced C as the industry's high-performance systems-implementation language of choice. However, C programming continues to be an important and valuable skill, because of the enormous amount of C legacy code that must be maintained in industry. We point out pitfalls and explain procedures for dealing with them effectively. Students are highly motivated by the fact that they are learning a leading-edge language (C++) and a leading-edge programming paradigm (object-oriented programming) that will be immediately useful to them as they leave the college environment.

*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—Microsoft Visual C++ 6, Microsoft Visual C++ .NET, two versions of Borland C++Builder and two versions of GNU C++. For the most part, the programs in the text will work on all ANSI/ISO standard-compliant compilers; we posted the few problems we found at **www.deitel.com**. When possible, we also posted the exact fixes required to enable those programs to work with a particular compiler.

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 example 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 and is the focus of our multimedia *Cyber Classrooms* and Web-based training courses. 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 C++ learning experience.

*Objectives*
Each chapter begins with objectives that inform students of what to expect and gives them an opportunity, after reading the chapter, to determine whether they have met the intended objectives. The objectives serve as confidence builders.

*Quotations*
The chapter objectives are followed by sets of quotations. Some are humorous, some are philosophical and some offer interesting insights. We have found that students enjoy relating the quotations to the chapter material. Many of the quotations are worth a second look *after* you read the chapters.

*Outline*
The chapter outline enables students to approach the material in a top-down fashion. Along with the chapter objectives, the outline helps students anticipate future topics and set a comfortable and effective learning pace.

*20,704 Lines of Syntax-Colored Code in 267 Example Programs (with Program Outputs)*
We present C++ features in the context of complete, working C++ programs. These LIVE-CODE programs range in size from just a few lines of code to substantial examples containing several hundred lines of code. Each program is 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 back to the program statements that produce those outputs is an excellent way to learn and to reinforce concepts. Our programs exercise the diverse features of C++. The code is syntax colored with C++ keywords, comments and other program text each appearing in different colors. This facilitates reading the code—students especially will appreciate the syntax coloring when they read the larger programs we present. All of the examples are available on the book's CD and are free for download at **www.deitel.com**.

*598 Illustrations/Figures*
An abundance of charts, line drawings and program outputs is included. We have converted all flowcharts to UML activity diagrams. We also use UML class diagrams in Chapters 9, 10, 12, 14 and 22 to model the relationships between classes throughout the text.

*601 Programming Tips*
We have included six types of programming tip to help students focus on important aspects of program development, testing and debugging, performance and portability. We highlight hundreds of these tips as *Good Programming Practices*, *Common Programming Errors*, *Performance Tips*, *Portability Tips*, *Software Engineering Observations* and *Testing and Debugging Tips*. These tips and practices represent the best we could glean from almost six decades (combined) of programming and teaching experience. One of our stu-

dents—a mathematics major—told us recently that she feels this approach is similar to the highlighting of axioms, theorems and corollaries in mathematics books, because it provides a sound basis on which to build good software.

### 90 Good Programming Practices

Good Programming Practices *are tips that call attention to techniques that help students produce programs that are more readable, self-documenting and easier to maintain. When we teach introductory courses to nonprogrammers, we state that the "buzzword" of each course is "clarity," and we tell the students that we will highlight (in these* Good Programming Practices*) techniques for writing programs that are clearer, more understandable and more maintainable.*

### 198 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 *reduces the likelihood that students will makes the same mistakes. It also shortens long lines outside instructors' offices during office hours!*

### 88 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 ways. Students really care about performance and they want to know what they can do to produce the most efficient programs. So we include* Performance Tips *that highlight opportunities for improving program performance—making programs run faster or minimizing the amount of memory that they occupy.*

### 36 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. Some programmers assume that if they implement an application in standard C++, the application will be portable. This is simply not the case. Achieving portability requires careful and cautious design. There are many pitfalls. We include* Portability Tips *to help students write portable code and to provide insights on how C++ achieves its high degree of portability.*

### 149 Software Engineering Observations

*The object-oriented programming paradigm necessitates a complete rethinking of the way we build software systems. C++ is an effective language for achieving good software engineering. The* Software Engineering Observations *highlight architectural and design issues, that affect the 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.*

### 38 Testing and Debugging Tips

*When we first designed this "tip type," we thought the tips would contain suggestions strictly for exposing bugs and removing them from programs. In fact, many of the tips describe as-*

*pects of C++ that prevent "bugs" from getting into programs in the first place, thus simplifying the testing and debugging process.*

***Summary (875 Summary bullets)***
Each chapter ends with additional pedagogical devices. We present a thorough, bullet-list-style summary of the chapter. This helps the student review and reinforce key concepts. There is an average of 40 summary bullets per chapter.

***Terminology (1782 Terms)***
We include an alphabetized list of the important terms defined in the chapter in a *Terminology* section. Again, this serves as further reinforcement. There are, on average, 81 terms per chapter. Each term also appears in the index, so the reader can locate terms and definitions quickly.

***555 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.

***800 Exercises (Solutions in Instructor's Manual; Count Includes Separate Parts)***
Each chapter concludes with a substantial set of exercises including simple recall of important terminology and concepts; writing individual C++ statements; writing small portions of C++ functions and classes; writing complete C++ functions, 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. The solutions for the exercises are included on the *Instructor's CD* which 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 ancillary is limited strictly to college professors teaching from the book. Instructors may obtain the solutions manual only from their Prentice Hall representatives.]** Students and professional readers can obtain solutions to approximately half the exercises in the book by purchasing the optional *C++ Multimedia Cyber Classroom, 4/e*. The *Cyber Classroom* offers many other valuable capabilities as well and is ideal for self study and reference. Also available is the boxed product, *The Complete C++ Training Course, 4/e*, which includes both our textbook, *C++ How to Program, 4/e*, and the *C++ Multimedia Cyber Classroom, 4/e*. All of our *Complete Training Course* products are available at bookstores and online booksellers, including **www.informIT.com**.

***Approximately 5,000 Index Entries (with approximately 7,700 Page References)***
We have included an extensive *Index* at the back of the book. Using this resource, readers can search for any term or concept by keyword. The *Index* is useful to people reading the book for the first time and is especially useful to professional programmers who use the book as a reference. These index entries also appear as hyperlinks in the *C++ Multimedia Cyber Classroom, 4/e*.

***"Double Indexing" of All C++ LIVE-CODE Examples***
*C++ How to Program, 4/e*, has 267 LIVE-CODE examples, which we have "double indexed." For every C++ source-code program in the book, we took the figure caption and in-

dexed it both alphabetically and as a subindex item under "Examples." This makes it easier to find examples that are demonstrating particular features. Each of the figure captions also appears in the Illustrations section (following the Contents section) at the front of the book.

## *Software Included with* C++ How to Program, 4/e

*C++ How to Program, 3/e,* included on its CD the Microsoft Visual C++ 6 Introductory Edition development environment. In *C++ How to Program, 4/e,* we wanted to include Microsoft's new Visual C++ .NET development environment, but Microsoft was not as yet making this software available to be included with textbooks. As soon as Microsoft does make Visual C++ .NET available, we will post information at our Web site indicating how students and professionals can obtain this software; there will be separate instructions for students and professionals. *C++ How to Program, 4/e*, includes Microsoft Visual C++ 6 Introductory Edition. A separate value-pack option also is available that contains Metrowerks CodeWarrior (ISBN# 0-13-101151-0); for more information on this option please write to **cs@prenhall.com** or **deitel@deitel.com**.

*Free C++ Compilers and Trial-Edition C++ Compilers on the Web*

This section overviews C++ compilers that are available for download over the Web. We discuss only those compilers that are available for free or as free-trial versions. Please keep in mind that in many cases, the trial-edition software cannot be used after the trial period has expired.

One popular organization that develops free software is the GNU Project (**www.gnu.org**), originally created to develop a free operating system similar to UNIX. GNU offers developer resources, including editors, debuggers and compilers. Many developers use the gcc (GNU Compiler Collection) compilers, available for download from **gcc.gnu.org**. This product contains compilers for C, C++, Java and other languages. The gcc compiler is a command-line compiler (i.e., it does not provide a graphical user interface). Many Linux and UNIX systems come with the gcc compiler installed. Red Hat has developed Cygwin (**www.cygwin.com**), an emulator that allows developers to use UNIX commands on Windows. Cygwin includes the gcc compiler.

Intel provides 30-day trial versions for its Windows and Linux C++ command-line compilers. The 30-day trial period also includes free customer support. Information on both compilers can be found at **developer.intel.com/software/products/global/eval.htm**.

Borland provides a Windows-based C++ developer product called C++Builder (**www.borland.com/cbuilder/cppcomp/index.html**). The basic C++Builder compiler (a command-line compiler) is free for download. Borland also provides several versions of the C++Builder that contain graphical user interfaces (GUIs). These GUIs are more formally called integrated development environments (IDEs), and, unlike command-line compilers, enable the developer to edit, debug and test programs quickly. Using an IDE, many of the tasks that involved tedious commands can now be executed via menus and buttons. Some of these products are available on a free-trial basis. For more information on C++Builder, visit

**www.borland.com/products/downloads/download_cbuilder.html**

For Linux developers, Borland provides the Borland Kylix development environment. The Borland Kylix Open Edition, which includes an IDE, can be downloaded from

**www.borland.com/products/downloads/download_kylix.html**

Many of the downloads available from Borland require users to register.

The Digital Mars C++ Compiler (**www.digitalmars.com**), is available for Windows and DOS, and includes tutorials and documentation. Readers can download a command-line or IDE version of the compiler. The DJGPP C/C++ development system is available for computers running DOS. DJGPP stands for DJ's GNU Programming Platform, where DJ is for DJ Delorie, the creator of DJGPP. Information on DJGPP can be found at **www.delorie.com/djgpp**. Locations where the compiler can be downloaded at are provided at **www.delorie.com/djgpp/getting.html**.

***DIVE-INTO Series Tutorials for Popular C++ Environments***

We have launched our new *DIVE-INTO SERIES* of tutorials to help our readers get started with many popular C++ 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.)

Each of these tutorials shows how to compile, execute and debug C++ applications in that particular compiler product. Many of these documents also provide step-by-step instructions with screenshots to help readers to 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 the *Instructor's Manual* with solutions to the vast majority of the end-of-chapter exercises and a *Test Item File* of multiple-choice questions (approximately two per book section). 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
- Example source code
- 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*, *Software Engineering Observations* and *Testing and Debugging 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**.

## C++ in the Lab

This lab manual (full title: *C++ in the Lab, Lab Manual to Accompany C++ How to Program, Fourth Edition*; ISBN 0-13-038478-X) complements *C++ How to Program, 4/e*, and the optional *C++ Multimedia Cyber Classroom, 4/e*, by providing a series of hands-on lab assignments designed to reinforce students' understanding of lecture material. This lab manual is designed for closed laboratories, which are regularly scheduled classes supervised by an instructor. Closed laboratories provide an excellent learning environment because students can use concepts presented in class to solve carefully designed lab problems. Instructors are better able to gauge the students' understanding of the material by monitoring the students' progress in lab. This lab manual also can be used for open laboratories, homework and for self-study.

*C++ in the Lab* focuses on Chapters 1–14 and 17 of *C++ How to Program, 4/e*. Each chapter in the lab manual is divided into *Prelab Activities*, *Lab Exercises* and *Postlab Activities*.[3] Each chapter contains objectives that introduce the lab's key topics and an assignment checklist that allows students to mark which exercises the instructor has assigned. Each page in the lab manual is perforated, so students can submit their answers (if required).

3. We expect few introductory classes to advance beyond Chapter 10 of this lab manual. For this reason, the labs in Chapters 11–14 and 17 do not contain the extensive sets of activities available in the previous chapters. Nevertheless, instructors will be able to conduct effective labs using the exercises we have included on these more complex topics. Instructors with special requirements should write to **deitel@deitel.com**.