教育部高等教育司推荐
国外优秀信息科学与技术系列教学用书

C++程序设计

（第三版 影印版）

PROGRAMMING IN C++

(Third Edition)

■ Nell Dale
Chip Weems

高等教育出版社
Higher Education Press

C++ 程序设计

PROGRAMMING IN C++

教育部高等教育司推荐
国外优秀信息科学与技术系列教学用书

C++程序设计

（第三版 影印版）

PROGRAMMING IN C++

(Third Edition)

Nell Dale
Chip Weems

高等教育出版社
HIGHER EDUCATION PRESS

图字：01-2005-4685 号

Programming in C++, Third Edition

Nell Dale,Chip Weems

ORIGINAL ENGLISH LANGUAGE EDITION PUBLISHED BY

Jones and Bartlett Publishers, Inc.

40 Tall Pine Drive

Sudbury, MA01776

图书在版编目(CIP)数据

C++ 程序设计 = Programming in C++：第 3 版 / (美)戴利(Dale,N.),(美)维姆斯(Weems,C.)
影印本. -北京：高等教育出版社,2006.3
ISBN 7-04-019109-1

Ⅰ. C… Ⅱ. ①戴… ②维… Ⅲ. C 语言－程序设计－英文 Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2006)第 022030 号

出版发行	高等教育出版社	**购书热线**	010－58581118
社　　址	北京市西城区德外大街 4 号	**免费咨询**	800－810－0598
邮政编码	100011	**网　　址**	http://www.hep.edu.cn
总　　机	010－58581000		http://www.hep.com.cn
		网上订购	http://www.landraco.com
经　　销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印　　刷	北京民族印刷厂	**畅想教育**	http://www.widedu.com
开　　本	787×1092 1/16	**版　　次**	2006 年 3 月第 1 版
印　　张	46.75	**印　　次**	2006 年 3 月第 1 次印刷
字　　数	720 000	**定　　价**	45.00 元

物料号 19109-00

前　言

20世纪末，以计算机和通信技术为代表的信息科学和技术对世界经济、科技、军事、教育和文化等产生了深刻影响。信息科学技术的迅速普及和应用，带动了世界范围信息产业的蓬勃发展，为许多国家带来了丰厚的回报。

进入21世纪，尤其随着我国加入WTO，信息产业的国际竞争将更加激烈。我国信息产业虽然在20世纪末取得了迅猛发展，但与发达国家相比，甚至与印度、爱尔兰等国家相比，还有很大差距。国家信息化的发展速度和信息产业的国际竞争能力，最终都将取决于信息科学技术人才的质量和数量。引进国外信息科学和技术优秀教材，在有条件的学校推动开展英语授课或双语教学，是教育部为加快培养大批高质量的信息技术人才采取的一项重要举措。

为此，教育部要求由高等教育出版社首先开展信息科学和技术教材的引进试点工作。同时提出了两点要求，一是要高水平，二是要低价格。在高等教育出版社和信息科学技术引进教材专家组的努力下，经过比较短的时间，第一批引进的20多种教材已经陆续出版。这套教材出版后受到了广泛的好评，其中有不少是世界信息科学技术领域著名专家、教授的经典之作和反映信息科学技术最新进展的优秀作品，代表了目前世界信息科学技术教育的一流水平，而且价格也是最优惠的，与国内同类自编教材相当。

这项教材引进工作是在教育部高等教育司和高教社的共同组织下，由国内信息科学技术领域的专家、教授广泛参与，在对大量国外教材进行多次遴选的基础上，参考了国内和国外著名大学相关专业的课程设置进行系统引进的。其中，John Wiley公司出版的贝尔实验室信息科学研究中心副总裁Silberschatz教授的经典著作《操作系统概念》，是我们经过反复谈判，做了很多努力才得以引进的。William Stallings先生曾编写了在美国深受欢迎的信息科学技术系列教材，其中有多种教材获得过美国教材和学术著作者协会颁发的计算机科学与工程教材奖，这批引进教材中就有他的两本著作。留美中国学者Jiawei Han先生的《数据挖掘》是该领域中具有里程碑意义的著作。由达特茅斯学院Thomas Cormen和麻省理工学院、哥伦比亚大学的几

位学者共同编著的经典著作《算法导论》，在经历了 11 年的锤炼之后于 2001 年出版了第二版。目前任教于美国 Massachusetts 大学的 James Kurose 教授，曾在美国三所高校先后 10 次获得杰出教师或杰出教学奖，由他主编的《计算机网络》出版后，以其体系新颖、内容先进而倍受欢迎。在努力降低引进教材售价方面，高等教育出版社做了大量和细致的工作。这套引进的教材体现了权威性、系统性、先进性和经济性等特点。

教育部也希望国内和国外的出版商积极参与此项工作，共同促进中国信息技术教育和信息产业的发展。我们在与外商的谈判工作中，不仅要坚定不移地引进国外最优秀的教材，而且还要千方百计地将版权转让费降下来，要让引进教材的价格与国内自编教材相当，让广大教师和学生负担得起。中国的教育市场巨大，外国出版公司和国内出版社要通过扩大发行数量取得效益。

在引进教材的同时，我们还应做好消化吸收，注意学习国外先进的教学思想和教学方法，提高自编教材的水平，使我们的教学和教材在内容体系上，在理论与实践的结合上，在培养学生的动手能力上能有较大的突破和创新。

目前，教育部正在全国 35 所高校推动示范性软件学院的建设和实施，这也是加快培养信息科学技术人才的重要举措之一。示范性软件学院要立足于培养具有国际竞争力的实用性软件人才，与国外知名高校或著名企业合作办学，以国内外著名 IT 企业为实践教学基地，聘请国内外知名教授和软件专家授课，还要率先使用引进教材开展教学。

我们希望通过这些举措，能在较短的时间，为我国培养一大批高质量的信息技术人才，提高我国软件人才的国际竞争力，促进我国信息产业的快速发展，加快推动国家信息化进程，进而带动整个国民经济的跨越式发展。

教育部高等教育司

二〇〇二年三月

Preface

The first two editions of *Programming in C++* have provided a straight-forward, no-frills introduction to C++. Throughout the successive editions of this book, one thing has not changed: our commitment to the student. As always, our efforts are directed toward making the sometimes difficult concepts of computer science more accessible to all students.

This edition of *Programming in C++* continues to reflect our philosophy that topics once considered too advanced can be taught in the first course. For example, we address metalanguages explicitly as the formal means of specifying programming language syntax. We discuss modular design in terms of abstract steps, concrete steps, functional equivalence, and functional cohesion. Preconditions and postconditions are used in the context of the algorithm walk-through, in the development of testing strategies, and as interface documentation for user-written functions. Data abstraction and abstract data types (ADTs) are explained in conjunction with the C++ class mechanism, creating a natural lead-in to object-oriented programming.

ISO/ANSI-standard C++ is used throughout the book, including relevant portions of the new C++ standard library.

Changes to the Third Edition

The third edition does not introduce additional C++ syntax nor change the order of the content. What we have done is completely revamp the goals, the case studies, and the exercises. In addition, beginning with Chapter 13, the language of the material has become more object oriented.

Goals The chapter goals have been reorganized to reflect two aspects of learning: knowledge and skills. Thus, the goals are divided into two sections. The first lists the knowledge goals, phrased in terms of what the student should know after reading the chapter. The second lists what the student should be able to do after reading the chapter.

Programming Examples Each chapter has a completely new programming example. Programming examples that begin with a problem statement and end with a tested program have been the hallmark of our books. In this edition, we have added screen shots of the output for each of the examples.

All of the exercises are new in this edition. The number of exercises has been expanded by between twenty and thirty percent. All of the programming problems are new.

Object-Oriented Language The List ADT in Chapter 13 has been changed by removing the Print operation and introducing an iterator pair, `Reset` and `GetNextItem`. This change provides better encapsulation. The list does not need to know anything about the items that it contains. The list simply returns objects to the client program, which should know what the objects are. The flaw in this design is pointed out in Chapter 14: The `Delete` and `BinSearch` operations use the relational operators, limiting the type of item to built-in types. In this chapter, the relational operators are replaced by operations `LessThan` and `Equal`; the documentation states that `ItemType` must implement these operations. The concepts of action responsibilities and knowledge responsibilities also are discussed. Each class is independently tested, stressing the importance of testing.

C++ and Object-Oriented Programming

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

It must be emphasized that although we use C++ as a vehicle for teaching computer science concepts, the book is not a language manual and does not attempt to cover all of C++. Certain language features—templates, operator overloading, default arguments, run-time type information, and mechanisms for advanced forms of inheritance, to name a few—are omitted in an effort not to overwhelm the beginning student with too much, too fast.

There are diverse opinions about when to introduce the topic of object-oriented programming (OOP). Some educators advocate an immersion in OOP from the very beginning, whereas others (for whom this book is intended) favor a more heterogeneous approach, in which both functional decomposition and object-oriented design are pre-

sented as design tools. The chapter organization of *Programming in C++* reflects a transitional approach to OOP. Although we provide an early preview of object-oriented design in Chapter 4, we delay a focused discussion until Chapter 14, after students have acquired a firm grounding in algorithm design, control abstraction, and data abstraction with classes.

Synopsis

Chapter 1 is designed to create a comfortable rapport between students and the subject. The basics of hardware and software are presented, and problem-solving techniques are introduced and reinforced in a problem-solving case study.

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

Functional decomposition and object-oriented design methodologies are a major focus of Chapter 4, and the discussion is written with a healthy degree of formalism. The treatment of object-oriented design this early in the book is more superficial than that of functional decomposition. However, students gain the perspective early that there are two—not just one—design methodologies in widespread use and that each serves a specific purpose. Object-oriented design is covered in depth in Chapter 14. Chapter 4 also covers input and file I/O. The early introduction of files permits the assignment of programming problems that require the use of sample data files.

Students learn to recognize functions in Chapters 1 and 2, and they learn to use standard library functions in Chapter 3. Chapter 4 reinforces the basic concepts of function calls, argument passing, and function libraries. Chapter 4 also relates functions to the implementation of modular designs, and begins the discussion of interface design that is essential to writing proper functions.

Chapter 5 begins with Boolean data, but its main purpose is to introduce the concept of flow of control. Selection, using If-Then and If-Then-Else structures, is used to demonstrate the distinction between physical ordering of statements and logical ordering. We also develop the concept of nested control structures. Chapter 5 concludes with a lengthy Testing and Debugging section that expands on the modular design discussion by introducing preconditions and postconditions. The algorithm walk-through and code walk-through are introduced as means of preventing errors, and the execution trace is used to find errors that may have made it into the code.

Chapter 6 is devoted to loop control strategies and looping operations using the syntax of the While statement. Rather than introducing multiple syntactical structures, our approach is to teach the concepts of looping using only the While statement. However, because many instructors have told us that they prefer to show students the syntax

for all of C++'s looping statements at once, the discussion of For and Do-While statements in Chapter 9 can be covered after Chapter 6.

By Chapter 7, students are already comfortable with breaking problems into modules and using library functions, and they are receptive to the idea of writing their own functions. Thus Chapter 7 focuses on passing arguments by value and covers flow of control in function calls, arguments and parameters, local variables, and interface design. Coverage of interface design includes preconditions and postconditions in the interface documentation, control abstraction, encapsulation, and physical versus conceptual hiding of an implementation. Chapter 8 expands the discussion to include reference parameters, scope and lifetime, stubs and drivers, and more on interface design, including side effects.

Chapter 9 covers the remaining "ice cream and cake" control structures in C++ (Switch, Do-While, and For), along with the Break and Continue statements. These structures are useful but not necessary. Chapter 9 is a natural ending point for the first quarter of a two-quarter introductory course sequence.

Chapter 10 begins the transition between the control structures orientation of the first part of the book and the abstract data type orientation of the second part. We examine the built-in simple data types in terms of the set of values represented by each type and the allowable operations on those values. We introduce additional C++ operators and discuss at length the problems of floating-point representation and precision. User-defined simple types, user-written header files, and type coercion are among the other topics covered in this chapter.

We begin Chapter 11 with a discussion of simple versus structured data types. We introduce the record (struct in C++) as a heterogeneous data structure, describe the syntax for accessing its components, and demonstrate how to combine record types into a hierarchical record structure. From this base, we proceed to the concept of data abstraction and give a precise definition to the notion of an ADT, emphasizing the separation of specification from implementation. The C++ class mechanism is introduced as a programming language representation of an ADT. The concepts of encapsulation, information hiding, and public and private class members are stressed. We describe the separate compilation of program files, and students learn the technique of placing a class's declaration and implementation into two separate files: the specification (.h) file and the implementation (.cpp) file.

In Chapter 12, the array is introduced as a homogeneous data structure whose components are accessed by position rather than by name. One-dimensional arrays are examined in depth, including arrays of structs and arrays of class objects. Material on multidimensional arrays completes the discussion.

Chapter 13 integrates the material from Chapters 11 and 12 by defining the list as an ADT. Because we have already introduced classes and arrays, we can clearly distinguish between arrays and lists from the beginning. The array is a built-in, fixed-size data structure. The list is a user-defined, variable-size structure, represented in this chapter as a length variable and an array of items bound together in a class object. The elements in the list are those elements in the array from position 0 through position *length* 2 1. In this chapter, we design C++ classes for unsorted and sorted list ADTs, and

we code the list algorithms as class member functions. Finally, we examine C strings in order to give students some insight into how a higher-level abstraction (a string as a list of characters) might be implemented in terms of a lower-level abstraction (a null-terminated char array).

Chapter 14 extends the concepts of data abstraction and C++ classes to an exploration of object-oriented software development. Object-oriented design, introduced briefly in Chapter 4, is revisited in greater depth. Students learn to distinguish between inheritance and composition relationships during the design phase, and C++'s derived classes are used to implement inheritance. This chapter also introduces C++ virtual functions, which support polymorphism in the form of run-time binding of operations to objects.

Chapter 15 concludes the text with coverage of recursion. There is no consensus as to the best place to introduce this subject. We believe that it is better to wait until at least the second semester to cover it. However, we have included this material for those instructors who have requested it. We suggest the following prerequisite reading for the topics in Chapter 15:

Section(s)	Topic	Prerequisite
15.1–15.3	Recursion with simple variables	Chapter 8
15.4	Recursion with arrays	Chapter 12

Additional Features

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

Programming Examples Problem solving is demonstrated through examples. In each example, we present a problem and use problem-solving techniques to develop a manual solution. Then we code the algorithm in C++. We show sample test data and output and follow up with a discussion of what is involved in thoroughly testing the program.

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

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

Exam Preparation Exercises These questions help the student prepare for tests. The questions usually have objective answers and are designed to be answerable with a few minutes of work. Answers to selected questions are given in the back of the book, and the remaining questions are answered in the *Instructor's Guide.*

Programming Warm-Up Exercises This section provides the student with experience in writing C++ code fragments. The student can practice the syntactic constructs in each chapter without the burden of writing a complete program. Solutions to selected questions from each chapter appear in the back of the book; the remaining solutions can be found in the *Instructor's Guide.*

Programming Problems These exercises, drawn from a wide range of disciplines, require the student to design solutions and write complete programs.

Programming Example Follow-Up Much of modern programming practice involves reading and modifying existing code. These exercises give the student an opportunity to strengthen this critical skill by answering questions about the example code or by making changes to it. All of the solutions to these exercises are in the *Instructor's Guide,* rather than the text, allowing the instructor the flexibility of assigning them as programming problems.

Supplements

Instructor's Guide and Test Bank The *Instructor's Guide* features chapter-by-chapter teaching notes, answers to the balance of the exercises, and a compilation of exam questions with answers. The *Instructor's Guide,* included on the *Instructor's ToolKit,* is available to adopters on request from Jones and Bartlett.

Instructor's ToolKit The *Instructor's ToolKit* is a powerful teaching tool available to adopters upon request from the publisher. It contains an electronic version of the *Instructor's Guide,* a computerized test bank, PowerPoint lecture presentations, and the complete programs from the text.

Programs The programs contain the source code for all of the complete programs that are found within the textbook. They are available on the *Instructor's ToolKit,* and also as a free download for instructors and students from the publisher's website (`http://computerscience.jbpub.com/cs_resources.cfm`). The programs from all of the programming examples, plus complete programs that appear in the chapter bodies, are included. (Fragments or snippets of program code are not included nor are the solutions to the chapter-ending "Programming Problems.") The program files can be viewed or edited using any standard text editor, but a C++ compiler must be used in order to compile and run the programs. The publisher offers compilers bundled with this text at a substantial discount.

Companion Website This website (`www.problemsolvingcpp.jbpub.com`) features the complete programs from the text.

A Laboratory Course in C++, Fourth Edition Written by Nell Dale, this lab manual follows the organization of this edition of the text. The lab manual is designed to allow the instructor maximum flexibility and may be used in both open and closed laboratory settings. Each chapter contains three types of activities: Prelab, Inlab, and Postlab. Each lesson is broken into exercises that thoroughly demonstrate the concepts covered in the corresponding chapter. The programs, program shells (partial programs), and data files that accompany the lab manual can be found on the website for this book (`www.problemsolvingcpp.jbpub.com`).

Acknowledgments

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

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

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

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

We also thank Mike and Sigrid Wile, along with the many people at Jones and Bartlett who contributed so much, especially Stephen Solomon and Anne Spencer. Our special thanks go to Amy Rose, our Production Manager, whose skills and genial nature turn hard work into pleasure.

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

N.D.
C.W.

CONTENTS