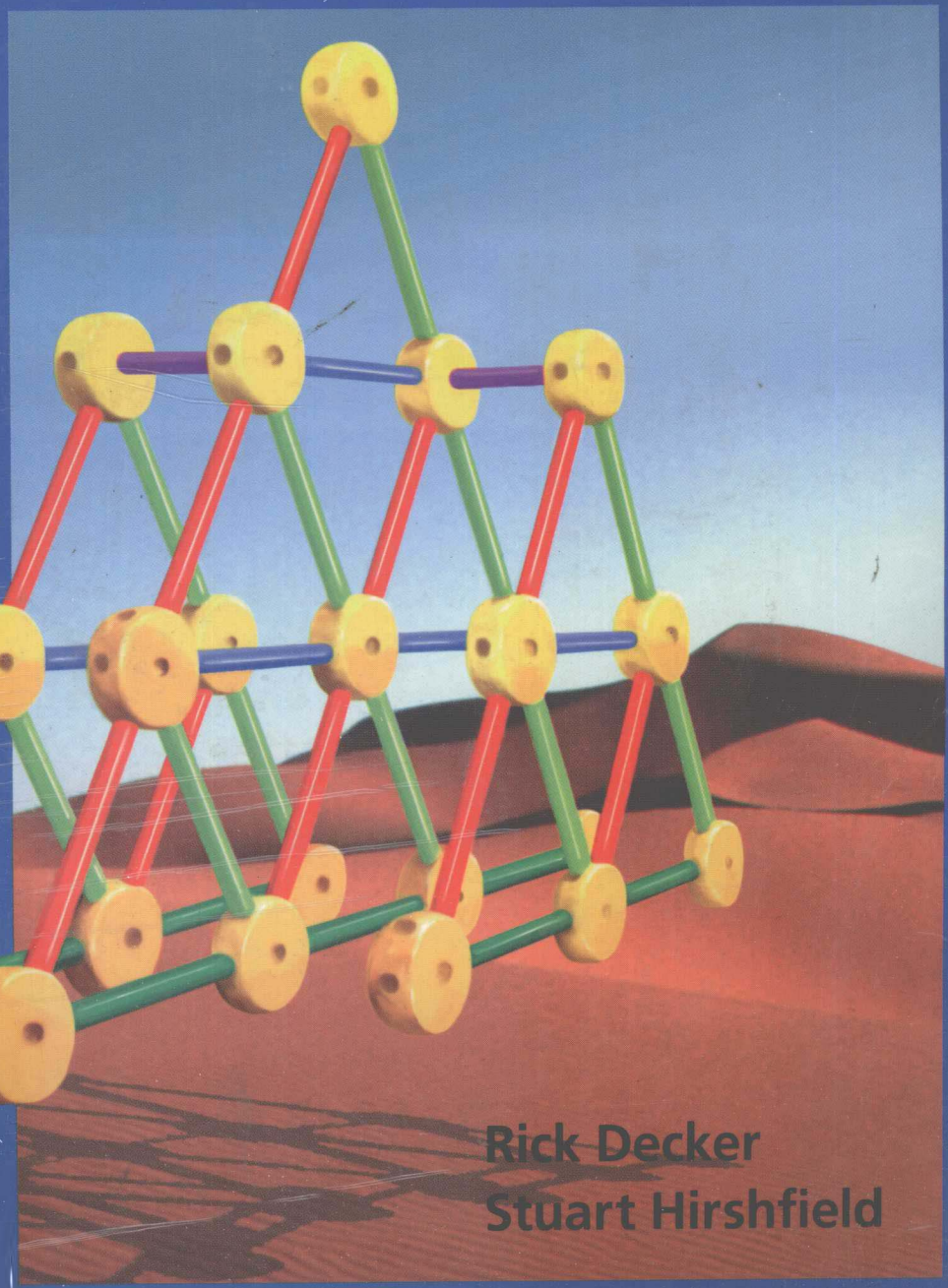
The Object Concept

AN INTRODUCTION TO COMPUTER PROGRAMMING USING C++



3.5" IBM PC Compatible Disk Enclosed

Rick Decker
Stuart Hirshfield

THE OBJECT CONCEPT

An Introduction to Computer Programming Using C++

RICK DECKER
STUART HIRSHFIELD
Hamilton College

PWS Publishing Company

I(T)P **International Thomson Publishing Company**

Boston • Albany • Bonn • Cincinnati • Detroit • London • Madrid
Melbourne • Mexico City • New York • Paris • San Francisco
Singapore • Tokyo • Toronto • Washington


PWS PUBLISHING COMPANY
20 Park Plaza, Boston, Massachusetts 02116-4324

Copyright © 1995 by PWS Publishing Company, a division of International Thomson Publishing Inc.

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means—electronic, mechanical, photocopying, recording, or otherwise—without the prior written permission of PWS Publishing Company.

I(T)P™ International Thomson Publishing
The trademark ITP is used under license.

For more information, contact:

PWS Publishing Co.
20 Park Plaza
Boston, MA 02116

Nelson Canada
1120 Birchmount Road
Scarborough, Ontario
Canada M1K 5G4

International Thomson Publishing GmbH
Königswinterer Strasse 418
53227 Bonn, Germany

International Thomson Publishing Europe
Berkshire House I68–I73
High Holborn
London WC1V 7AA England

Thomas Nelson Australia
102 Dodds Street
South Melbourne, 3205
Victoria, Australia

International Thomson Publishing Asia
221 Henderson Road
#05–10 Henderson Building
Singapore 0315

International Thomson Publishing Japan
Hirakawacho Kyowa Building, 31
2-2-1 Hirakawacho
Chiyoda-ku, Tokyo 102
Japan

International Thomson Editores
Campos Eliseos 385, Piso 7
Col. Polanco
11560 Mexico D.F., Mexico

Library of Congress Cataloging-in-Publication Data
Decker, Rick.
The object concept : an introduction to computer programming using C++/Rick Decker, Stuart Hirshfield
p. cm.
Includes index.
ISBN 0-534-20496-1
1. C++ (Computer program language) 2. Object-oriented programming (Computer science) I. Hirshfield, Stuart. II. Title.
QA76.73.C153D43 1995 94-41156
005.13'3—dc20 CIP


This book is printed on recycled, acid-free paper.

Sponsoring Editor: Michael J. Sugarman
Developmental Editor: Mary Thomas
Production Editor: Abigail M. Heim
Marketing Manager: Nathan Wilbur
Manufacturing Coordinator: Lisa Flanagan
Editorial Assistant: Benjamin Steinberg

Interior Designer: Catherine Hawkes Design
Cover Designer: Julia Gecha
Cover Artist: Angela Perkins
Typesetter and Interior Illustrator: Pure Imaging
Cover Printer: New England Book Components
Text Printer and Binder: Quebecor Printing/Martinsburg

Cover Image: The TINKERTOY® product © 1993 by Playskool, Inc., a division of Hasbro, Inc. All rights reserved. Used with permission.

Printed and bound in the United States of America.
97 98 99—10 9 8 7 6 5

For Barb and Joanne

PREFACE

Motivation

Our dual goals in developing this text/lab package were (1) to render the concepts of object-oriented programming accessible and useful to novice programmers and (2) to do so in a manner that was coherent and meaningful to those of us who would be asked to teach these concepts. As such, the package can be viewed as representing a departure from traditional introduction to programming texts. We don't see it that way. Rather, it appears to us to be an evolutionary approach that applies common sense, established pedagogical techniques, and current software technology to the problems of teaching undergraduates to solve problems with computers. Further, it does so in a way that builds on all of our collective experience teaching programming.

Our own experience in recent years has been increasingly frustrating. Despite our best efforts (teaching Pascal; using highly interactive and supportive programming environments; incorporating hands-on laboratory experiences; providing students with interesting and complete sample programs to read, analyze, and experiment with; and so on), all but our very best students appeared to us to lack what are currently regarded as basic software engineering skills. That is, of those students who produce working programs, relatively few write programs that can be considered modular, readable, testable, and maintainable. Still fewer are capable—even after a typical CS1 course—of analyzing, specifying, designing, and managing even modest-sized programs of their own. Rhetoric and current programming texts notwithstanding, our students were not being trained as problem solvers and were not developing skills that we regarded as essential to both their subsequent course work and to their

此为试读,需要完整PDF请访问: www.ertongbook.com

careers. Our frustration has led us to reconsider what and how we are teaching novice programmers.

Object-Oriented Programming in CS1

This package reflects the position that the object-oriented paradigm is the one best suited for teaching introductory programming. Philosophically, this approach is justified by our feeling that the real value of the paradigm (and its significance to computer science education) is that it effectively raises the level of abstraction for all programmers, novices included. Who, after all, stands to benefit more from a higher-level programming interface than do novices?

From the more practical standpoint of course content, the decision to teach object-oriented programming (OOP) in CS1 is justified on the grounds that it extends the familiar procedural paradigm to effectively address all of the aforementioned frustrations. Generally speaking, OOP emphasizes a strategic, problem-solving approach to programming. Such an approach, in which design decisions are not only paramount, but are also clearly reflected in the resulting code, brings us a step closer to the idealized program development life cycle that most of us have been advocating for some years now. Indeed, OOP supports directly many of the software engineering concepts that are among the most difficult to convey in procedural terms: code reuse, encapsulation, incremental development and testing, and, of course, program design.

Our Approach

Programming texts have for many years been organized around the fundamental syntactic and semantic constructs supported by the language being taught. In the most recent past, these constructs were primarily procedural. As subprograms, for example, came to be regarded as important to effective programming, they were increasingly emphasized and, in many cases, moved to earlier chapters in texts. Data types and data abstraction were introduced as needed to support the development of more ambitious algorithms. We have applied similar thinking to teaching object-oriented programming. We have identified the fundamental, empowering constructs of the paradigm—those that support most directly the identification, creation, and use of high-level classes—and pushed them to the fore, essentially reversing the order of presentation of what are otherwise conventional CS1 topics. Algorithmic constructs are introduced in this context as a means to support class implementations.

What is distinctive about this approach is the fact that we do more than acknowledge the object-oriented paradigm—we embrace it. We focus from the outset on the object-oriented features of C++; that is, how classes are declared, defined, used, and organized into coherent designs. In the first part of the course, we concentrate on using classes as the basis for program specification

and design. Then, in the second part of the course, the predefined types of C++ are described in object-oriented terminology; that is, as related combinations of data, operators, and functions. The basic concepts of inheritance, construction, access control, and overloading are described in the third section. Thus, students are provided with both a framework and the building blocks with which they can define classes of their own, the primary activity of the final part of the course. Throughout the course, full-blown sample programs are used both to illustrate specific OOP and C++ features, and to allow students to interact with classes on a variety of levels.

The advantages of this approach to teaching novices are both numerous and tangible. First, introducing the object-oriented paradigm from the beginning allows us to exploit it as a design medium. Second, doing so puts the procedural paradigm (along with the ideas of top-down design and stepwise refinement) into a meaningful and useful problem-solving context. Third, it eliminates (at least, for the student!) the dreaded "paradigm shift" from procedural programming to object-oriented programming. Finally, and most important, it helps students to develop their problem-solving skills in conjunction with their programming skills.

Why C++?

A simplistic—and not totally irreverent—answer to the question of why we chose to write the book around C++ is "Why not?" If what we are hearing from industry and our students and what we are seeing at conferences is indicative, C is being replaced by its natural superset, C++, in the "real world." Truth be told, the pressure to teach C++ in the interest of better preparing our students for employment was, for us, a great reason *not* to use it as the language for this text. We succumbed—and have subsequently become converted—for a number of other reasons.

- First and foremost, C++ matches our interpretation of object-oriented programming. That is, just as the object-oriented paradigm extends the procedural one to incorporate user-defined classes, C++ is advertised as an extension of modern procedural languages with the features necessary to support classes.
- The fact that C++ is a hybrid language (not necessarily "purely" object-oriented) is also an advantage because it allows our (and some of our students') experience with algorithms and top-down design to come more directly into play than it might have had we chosen a "pure" OOP language.
- Most implementations of C++ support all of the modern software engineering concepts (for example, separately compilable files, incremental development and testing, and reusable code libraries) that we want students to take advantage of early in their programming experience.

- Although most of the reasons we had for adopting Pascal (over C) as the language for teaching CS1 some years ago still hold, C++ succeeds in overcoming most of what we regarded as the awkwardnesses of C (by, for example, providing reference parameters and loosening its dependence on the preprocessor). In short, we regard C++ as much more than "a better C." Based on its strengths as a design tool and its support of the entire software engineering life cycle, we consider it a better Pascal and thus an excellent choice for teaching CS1.

Pedagogy

Our approach to teaching CS1 has clearly changed, but the basic content and the goals of the course have not. It should not be surprising, then, that two of the pedagogical techniques that have proven most useful in teaching introductory programming have been adapted to the task of teaching object-oriented programming. First, our course is lab based. We provide students with detailed, directed, experimental laboratory exercises that help them explore firsthand the principles of OOP in a controlled fashion. Quoting from the preface of our (similarly lab-based) text, *Pascal's Triangle*, "These exercises are integrated precisely with the textual material and serve to bring the static text material to life. We've used a lab-based approach at our school for ten years for one main reason—it works."

In addition, each lab/chapter pair is based on a complete, working, motivating, and interesting sample program (a Program in Progress, or "PIP") written to illustrate particular language and OOP features and to provide a vehicle for experimentation in lab. Students are introduced to new concepts in the text. In the process of reading the text, students *read* the chapter's PIP, which is used to illustrate the new concepts. Then, in the laboratory they *use* the PIP they have read, *experiment* with it, and *extend* it using what they have learned from the text. With the exception of Chapter 1 (in which the students type in the PIP to gain practice with editing programs in their environment), all the PIPs are provided on the lab disk that accompanies the text.

The Details

Much about the organization of this package will look familiar to you. There are eleven chapters, each with a corresponding lab, roughly one for each week to fit a traditional semester once exams and review classes are figured in. We cover all of the traditional CS1 programming topics (admittedly, in an unconventional order) and wind up with presentations of algorithms and abstract data types—right where we want to be for CS2.

Conceptually, we have divided the text into four sections. The first section is devoted to providing students with the vocabulary and methodology needed to describe problems in object-oriented (or, more accurately, "class-oriented")

terms. **Chapter 1**, "Designing with Classes," describes briefly the evolution of and motivation for the object-oriented paradigm and provides some real-world examples of hierarchical systems. Next, we introduce the topic of program design and demonstrate how classes can effectively serve as a means for high-level program description. Finally, we introduce a simple method and notation, which we have dubbed the "Declare-Define-Use" approach, for relating program descriptions and C++ code. This early concentration on "description" helps students to focus on problem analysis and design without worrying about the details of implementation (which is particularly easy since students haven't seen C++ yet!). To be sure, we don't expect them to appreciate the coding details of the PIP for Chapter 1. On the other hand, the program (a stop-watch simulation) is conceptually simple enough that it can be read and used as we intended: to illustrate the correspondence between an object-oriented description of a problem and its more formal representation in C++.

Chapters 2 through 6 comprise the second section of the text. Collectively, these chapters describe the basic data types of C++ in formal, class-oriented terms. The goals are to convince students that class description is the fundamental activity of C++ programming and to establish both a problem-solving framework and a C++ vocabulary with which they can come to define their own classes. "The Ingredients of Classes," that is, primitive data types, operators, and simple functions, are introduced in **Chapter 2**. The Chapter 2 PIP is a fraction package that references both a standard and a user-defined library to illustrate the basic structure and organization of a C++ program.

Chapters 3 and 4 describe the basic algorithm control structures of C++ in the context of defining member functions. The PIPs for these chapters combine to form a single program that simulates a soda machine. The design and implementation of the machine itself, along with the C++ repertoire of selection statements, are presented in **Chapter 3**. The classes that support a general, menu-based interface are developed with the help of C++ repetition statements in **Chapter 4**. In both chapters, we emphasize how control structures fit into an object-oriented paradigm. Rather than being used as a basis for making high-level design decisions, they serve to express algorithmic details in a top-down fashion within the context of an object-oriented design.

Chapter 5, entitled "Compound Data," presents the composite classes of C++, including arrays, structures, and enumerations. Again, these concepts are rendered somewhat less intimidating by virtue of the fact that students are already familiar with the C++ notation for describing classes, a natural extension of structures. The PIP for this chapter, a card-playing program, demonstrates both the access operators for the composite classes and the language control structures that facilitate their use.

We discuss the topics of pointers and references in **Chapter 6**, each being described as a derived class. The dereferencing and address operators are illustrated, as are the allocation and deallocation functions . Arrays and strings are also described in pointer-based terminology. In presenting these topics, we concentrate exclusively on the notions of indirection and notation, leaving for

Chapter 11 the more complex uses of pointers in abstract data types. The chapter's PIP is a simple phone directory that makes use of dynamic allocation.

The third major section of the text spans Chapters 7–9 and describes user-defined classes in much the same way as the built-in classes of C++ are described in section two. The obvious difference is that in this section we have the predefined classes to work with, thus enabling us to define higher-level classes that reflect the real-world applications they are intended to model. As you would expect, these chapters tend to emphasize the more purely "object-oriented" features of C++.

Chapter 7, for example, is entitled "Process I: Organizing and Controlling Classes," and its PIP is an elevator simulation. The C++ mechanisms for controlling access to member data and functions (private, public, and friends) are described, as is its means for enforcing type-safe linkage. The reference to "process" in the chapter's title is not to be taken lightly. This is the first of two chapters (the other being Chapter 9) that is devoted primarily to the process of programming. This chapter focuses on program design and develops more fully our DDU approach and its relation to C++.

Class inheritance is the topic of **Chapter 8**. We illustrate, through a payroll program PIP, how one derives classes from others. This, in turn, motivates a discussion of how to further control access to member functions and data via protected and static descriptors, as well as a more thorough treatment of class (in particular, base class) constructors. The topics of heterogeneous lists and polymorphism also are introduced, as our PIP prints paychecks for a variety of employee types.

Chapter 9, "Process II: Working with Classes," reconsiders the program development process in light of the C++ experience students have now had. As such, the chapter serves as a recap of the first three sections of the text and as a natural concluding point for many traditional CS1 syllabi. The intention is to show how the DDU approach can be used with C++ to not only design and code programs, but also to test, debug, and maintain programs. The program in this case is a simple word processor that uses the standard stream and string libraries along with our own extended string package.

The theme of the fourth and final section is defining general classes (for example, our string library from Chapter 9) that support the description and implementation of application-oriented classes. This leads to natural discussions of algorithm analysis (as illustrated by a collection of functions that implement searching and sorting techniques in **Chapter 10**) and finally to abstract data types in **Chapter 11**. The final PIP introduces the topic of generic types in the context of a linked list package.

As with most texts, this one can be used in a variety of ways to accommodate your goals for CS1 and your academic calendar. At Hamilton, with our 13-week semester, we cover Chapters 1 through 9 in order, using about one or two weeks per chapter. We spend slightly more time on Chapters 2 and 3 (to make sure that students are comfortable with the basic data types and algo-

rithm control structures) and Chapter 9 (to emphasize solid program development skills), and devote whatever time remains to in-class exams and to covering either Chapter 10 or 11.

SUPPLEMENTARY MATERIAL In addition to the data disk (IBM PC compatible) included with this book, and the accompanying *Lab Manual*, an *Instructor's Manual* is available from the publisher. A Macintosh version of the data disk is also available from the publisher.

Peroration

So there you are—you hold in your hands the makings of a one-semester introduction to programming course suitable for use in CS1. It applies established pedagogical techniques to the task of teaching high-level problem -solving skills to novice programmers. It illustrates clearly via meaningful examples the utility and power of the object-oriented paradigm and encourages students to work like professional programmers right from the start of their programming careers. Finally, it does so in a way that supports the way most of us have been teaching introductory programming for many years, and it fits naturally with a standard computer science curriculum.

While this project was in many ways our creation, it would not exist in its present form were it not for the contributions of many talented and dedicated people. Our thanks go out, in parallel, to the following people for their insightful reviews:

Owen Astrachan
Duke University

Tom Bullock
University of Florida

Mark Ciampa
Volunteer State Community College

George Converse
Southern Oregon State College

Charles Dierbach
Towson State University

Linda Elliott
La Salle University

Frank Kelbe

Soheil Khajenoori
University of Central Florida

Stephen P. Leach
Florida State University

John A. N. Lee
Virginia Polytechnic Institute and State University

Daniel Ling
Okanagan University College

Robert Lipton
Pennsylvania State University, Schuylkill

此为试读,需要完整PDF请访问: www.ertongbook.com

Bruce Mabis
University of Southern Indiana

Nathaniel G. Martin
University of Rochester

Robert Noonan
College of William & Mary

Martin Osborne
Western Washington University

Frank Paiano
Southwestern Community College

Rich Pattis
University of Washington

Jim Slack
Mankato State University

John Stoneback
Moravian College

David B. Teague
Western Carolina University

Christian Vogeli
Grand Valley State University

Raymond F. Wisman
Indiana University, Southeast

Lynn R. Ziegler
St. John's University

Special thanks, also, to Mike Sugarman, Susan McCulley, Abby Heim, Frank Ruggirello, and Tammy Goldfeld, each of whom influenced the final product (and, in some cases, the authors) in some significant and positive way. Now, let's get on with it.

Rick Decker
Stuart Hirshfield

CONTENTS

此为试读,需要完整PDF请访问: www.ertongbook.com