Rick Mercer

COMPUTING FUNDAMENTALS WITH C++

ANSI/ISO Compliant

object-oriented programming & design

2nd edition

Franklin, Beedle & Associates

About This Edition of Computing Fundamentals with C++

This textbook is written for a first course in C++ programming and design. It has been used extensively in the introductory computer science course (CS1) and introductory courses for engineering and business majors. It is appropriate for students with no programming or design experience, or for students with programming experience in another language.

Features

Traditional Topics—Recognizes the relevance and validity of object-oriented programming and design while emphasizing traditional computing fundamentals.

Standard C++—Follows the standards set forth by the International Standards Organization (ISO).

Flexibility—Maintains the objects-early approach of the first edition, but now C++ classes can optionally be introduced later.

Gentle Objects-Early Approach—Use, modify, then implement classes.

Not Tied to a Specific System—No bias toward a particular operating system or compiler.

Object-Oriented Analysis and Design Case Studies—Contains two complete and accessible case studies featuring object-oriented analysis and design.

Algorithmic Patterns—Help for beginning programmers in designing algorithms around a set of common algorithm generalities.

Object-Oriented Design Heuristics—Integrates several of the guidelines for good design found in Arthur Riel's *Object-Oriented Design Heuristics*.

Extensively Tested in the Classroom and Lab—Incorporates suggestions from six years of development in the classroom.

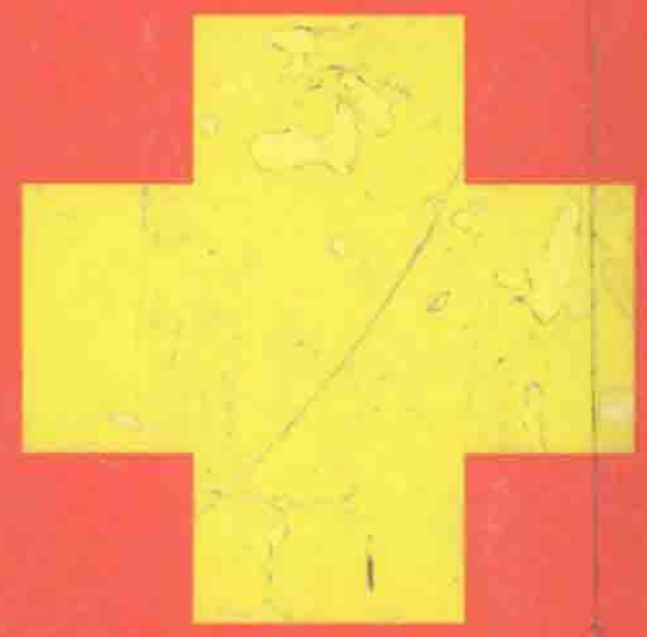
What's New in the Second Edition

This improved and expanded second edition includes these new chapters:

12: Object-Oriented Software Development: Analysis and Design
13: Object-Oriented Software Development: Design and Implementation
16: Object-Oriented Software Development: Inheritance and Polymorphism
18: Operator Overloading (from a first edition appendix)
20: Recursion

About the Author

Rick Mercer teaches computer science at the University of Arizona. In addition, he has been a guest lecturer at Duke University and Carnegie Mellon University. He has been invited to participate in numerous NSF-funded workshops concerning the implementation of object-oriented programming in the college computer science curriculum. This is the third textbook he has written for use in CS1. The first edition of *Computing Fundamentals with C++* was the number-one-selling C++ programming textbook in the CS1 college market.



Franklin, Beedle & Associates,

ISBN 1-887902-36-8

90000

9 781887 902366

9 8 7 6 5 4 3 2

Rick Mercer

COMPUTING FUNDAMENTALS WITH C++

ANSI/ISO Compliant

O
Ol
programming
& design

2nd edition

Franklin, Beedle & Associates, Inc.
8536 SW St. Helens Drive, Suite D
Wilsonville, Oregon 97070
503/682-7668
www.fbeedle.com

President & Publisher	Jim Leisy (jimleisy@fbeedle.com)
Production	Stephanie Welch
	Susan Skarzynski
	Tom Sumner
Manuscript Editor	Sheryl Rose
Developmental Editor	Sue Page
Marketing Group	Cary Crossland
	Carrie Widman
	Jason Smith
	Marc Chambers
	Stacia Houston
Order Processing	Chris Alarid
	Lois Allison
	Krista Hall

Library of Congress Cataloging-in-Publication Data

Mercer, Rick
Computing fundamentals with C++ : object-oriented programming & design / by Rick Mercer. -- 2nd ed.
p. cm.
Includes index.
ISBN 1–887902–36–8 (pbk.)
1. C++ (Computer program language) I. Title.
QA76.73.C153M46 1998
005.13'3--DC21 98-18880
CIP

Preface

This textbook is written for a first course in C++ programming and design. This book has been used extensively in the introductory computer science course (CS1) and introductory courses for engineering and business majors. It is appropriate for students with no programming or design experience, or for students with programming experience in another language.

This textbook emphasizes traditional computing fundamentals while recognizing the relevance and validity of object-oriented programming and design. Students completing the first 10 chapters will be comfortable with structured programming, vectors, and using standard C++ classes. They will also have the opportunity to implement some classes that have already been designed. After this, there are many options to choose for the first course.

I prefer to cover Chapters 1 through 13 during the introductory computer science course. This introduces problem solving, programming, and design using a team experience in object-oriented analysis and design (Chapters 12 and 13). I then use some of the remaining seven chapters to support my second course textbook (in CS2). The bonus chapters and built-in flexibility accommodate a range of preferences.

You have the option to cover classes early (before control structures) or later (after vectors). If presenting objects later, you may instead present matrix objects, recursion, and/or pointers with dynamic memory management.

This textbook is the result of 10 years of reasoning about how best to use the first year of the computer science curriculum and how best to integrate object technology into it. I believe that this process will continue for quite some time.

—*Rick H. Mercer*
http://www.cs.arizona.edu/people/mercer/compfun2/
email: **mercer@cs.arizona.edu**

Features

Traditional Topics. This textbook recognizes the relevance and validity of object-oriented programming and design while it emphasizes traditional computing fundamentals. The book's breadth of topics and flexibility allow for presentation of other traditional topics such as pointers and recursion. It also presents some C++ features that could well become traditional topics during the first two or three courses, such as templates for generic classes, operator overloading, and standard containers with iterators.

Standard C++. Because the International Standards Organization (ISO) has approved the C++ standard document, students can now study C++ as a language that has an internationally accepted standard. Many compilers now conform to the standard or soon will. This textbook, and I suspect most other new ones, will use the standard `string` and `vector` classes and the operations that work on any standard compiler. Your compiler may not yet be able to handle a few of these standards, but these can be easily fixed with one or two differences from the standard approach used in this text. At the time of this writing, you may have to slightly modify what you do with namespaces and the `#include` directives, but eventually the compilers will catch up. On older compilers, you might need to use the `string` and `vector` classes that come with this textbook.

Flexibility. This second edition maintains the objects-early approach of the first, but now C++ classes can optionally be introduced earlier (Chapter 6). Several programming projects and a few sections identified with a Chapter 6 prerequisite should be postponed until after presenting Chapter 6, "Class Definitions and Member Functions." You may select from a wide array of additional topics after Chapter 10, "Vectors." These include container classes and iterators (Chapter 11), indirection and dynamic memory allocation (Chapters 14 and 15), inheritance (Chapter 16), templates (Chapter 17), operator overloading (Chapter 18), doubly subscripted objects (Chapter 19), recursion (Chapter 20), and object-oriented analysis and design (Chapters 12 and 13).

Gentle Objects-Early Approach: Use, Modify, then Implement Classes. This second edition begins by placing the student in the role of a consumer—-you begin by using existing objects while honing problem-solving and program-development skills. You will then modify, enhance, and ultimately design and implement your own classes of increasing complexity.

Carefully Chosen Subset of Analysis, Design, and C++. Because students using this textbook might have little or no programming or design experience, several C++ features and subtleties are not presented. Students concentrate on a solid subset of this feature-

rich language. Some of C++'s trickier topics are delayed until the later chapters; for example, using the standard `string` class avoids early coverage of pointers.

Not Tied to a Specific System. There is little bias toward a particular operating system or compiler. However, this textbook presents `#includes` and namespaces according to the C++ standard. All other material applies to any computer system using standard C++. All software has been tested on the following systems: Borland C++ version 5.02; Microsoft Visual C++ version 5.0; Metrowerks Codewarrior (for both Macintosh and Windows); and GNU g++ 2.90.27 in a Unix environment.

Object-Oriented Analysis and Design Case Studies. This textbook is unique among first course textbooks in that it contains two complete and accessible case studies featuring object-oriented analysis and design. The cashless jukebox of Chapters 12 and 13 allows students to experience more than just programming language features and typing at the computer. Students work in teams as they analyze and design object-oriented solutions. This allows progress as students make design decisions about their systems. End-of-chapter projects ask for specific analysis and design deliverables:

- Role playing to indicate an understanding of the system.
- A set of CRH cards to capture analysis and design decisions.
- The C++ class definitions that represent the design.

If teams are not possible, or for those reading this textbook on their own, Chapters 12, 13, and 16 are written in an interesting narrative form to capture real-life team analysis and design experiences.

Another object-oriented analysis and design case study—the college library system—introduces students to the concepts of inheritance and polymorphism. Additionally, object-oriented design heuristics provide insights into the proper object decomposition and classification of systems. This edition emphasizes design.

Algorithmic Patterns. Algorithmic patterns help beginning programmers design algorithms around a set of common algorithm generalities. The first algorithmic pattern, and perhaps one of the oldest—Input/Process/Output (IPO)— is introduced in Chapter 1. It is reused in subsequent chapters. The IPO pattern is especially useful to students with no programming experience and to the lab assistants helping them. Other algorithmic patterns introduced in the appropriate places include Guarded Action and Determinate Loop.

Object-Oriented Design Heuristics. In his book *Object-Oriented Design Heuristics,* Arthur Riel catalogs about 60 guidelines to good design. This textbook integrates several of these guidelines in context. Students can make informed decisions that result in better designs.

Extensively Tested in the Classroom and Lab. This textbook was six years in the making. Students supplied many useful comments and suggestions concerning manuscript clarity, organization, projects, and examples. The tremendous personal contact and testability was made possible with closed lab sections for all students. This edition has been class tested over six consecutive terms, from summer 1996 to spring 1998 in four slightly different courses at Pennsylvania State University. The first edition also was extensively class tested over a three-year period.

Pedagogy

This textbook has many pedagogical features that make this introduction to programming, design, and object technology accessible to students.

Self-Check Questions. These short questions and answers allow students to evaluate whether they understand the details and terms presented in the reading. The answers to all self-check questions are included in the appendix at the back of this book.

Exercises. These transitional problems examine the major concepts presented in the chapter. Answers are in the instructor's manual and at this textbook's Web site in order to encourage students to write down the answers with paper and pencil, as if it were a practice test.

Programming Tips. Each set of weekly programming projects is preceded by a set of programming tips intended to help students complete programs, warn of potential pitfalls, and promote good programming habits.

Programming Projects. Many relatively small-scale problems have been extensively lab tested to ensure that projects can be assigned and completed with little or no instructor intervention. The programming projects are strategically positioned to occur every week of lecture to reinforce the concepts just presented. Chapter 7, "Selection," Chapter 8, "Repetition," and Chapter 10, "Vectors," all have two sets of exercises, programming tips, and programming projects.

Analysis and Design Projects. Chapters 1, 12, and 13 have analysis and design projects that allow students to participate in other aspects of program and software development. Chapter 1 projects allow students to begin by doing something in the lab without the computer. The projects in Chapters 12 and 13 allow students to participate in team projects at the analysis and design phases rather than implementation.

Scaffolding. Scaffolding is a teaching and learning technique that provides support for new concepts. The scaffolding is slowly removed as more is asked from students. Scaffolding is used in these ways:

- Applications presented in the chapters solve problems similar to some of the programming projects—this allows students to generalize on specific examples before implementing solutions to problems from different domains.
- Example dialogues allow for easier identification of input and output requirements—especially in the early chapters. This prepares students for designing their own input and output requirements later in the book.
- Students are initially given function headings and test drivers so they need to complete only the function body; later on, projects ask students to create all three items.
- Test drivers are provided (many on disk) for testing functions; later in the book, students must write their own test drivers.
- In the early chapters, students are given a class definition and some member functions, and are then asked to add other member functions to a class. Later on, students implement all member functions given the class definition.
- Students are first asked to implement member functions specified in the design (the C++ class definition). In Chapters 12 and 13, students analyze a problem and design class definitions before implementing member functions.
- Students begin by analyzing small problems; by Chapter 13 students analyze small systems.

What's New in the Second Edition

This improved and expanded second edition has taken almost as much time to complete as the first. The major changes deal with integrating design and introducing students to object-oriented software development. For example, to help with algorithm design, this edition presents algorithmic patterns such as Input/Process/Output and Guarded Action. These are particularly helpful to students with no experience as they reason about algorithm design. Object-oriented design heuristics such as 6.1, "All data should be hidden within a class," and 12.1, "Model the real world whenever possible," help students think about the many design decisions to make during software development. Several object-oriented design heuristics and two design patterns, State Object and Iterator, guide students through algorithmic patterns.

These new chapters have been added:

12: Object-Oriented Software Development: Analysis and Design

13: Object-Oriented Software Development: Design and Implementation

16: Object-Oriented Software Development: Inheritance and Polymorphism

18: Operator Overloading (from a first edition appendix)

20: Recursion

Additional changes include, but are not limited to the following:

- The standard `vector` class is presented before primitive C arrays.
- The `grid` class (inspired by Rich Pattis's "Karel the Robot") replaces the `ATM` and `bank` classes as an author-supplied class that presents new concepts over several chapters.
- New programming projects have been added.
- The standard list and iterator objects are presented as examples of a container object and the means to iterate over the elements.
- Classes are covered earlier (Chapter 6), or could be postponed until after control structures, functions/parameters, and vectors.
- A linked list class has been added.
- Chapter size has been reduced to allow coverage at the rate of one chapter per week.

Instructor's Manual

The instructor's manual is available on a disk obtained from the publisher, Franklin, Beedle & Associates. It contains the following:

- Solutions to all programming projects.
- Several sample tests for each chapter.
- Chapter-by-chapter suggestions.
- Answers to all exercises.
- The directory of files (also at this textbook's Web site).
- Slide shows for every chapter (also at this textbook's Web site).

World Wide Web Site

http://www.cs.arizona.edu/people/mercer/compfun2/

email: **mercer@cs.arizona.edu**

Acknowledgments

Critical feedback from students and other instructors is essential to creating a solid textbook. I have been fortunate enough to have small lecture sizes (10 to 35 students) and to be in all labs with all of my students for the past 10 years. This has enabled me to keep track of their progress and of their problems, which has dramatically helped produce a textbook that is accessible to the intended audience. I acknowledge and thank the students at Penn State Berks in past years and those students in the following courses that class tested this second edition: CSE 103 (fall '96 and fall '97), CmpSc 201 (fall '96, spring '97, fall '97, and spring '98), CmpSc 101 (the summers of '96 and '97), and CmpSc 203 (the summers of '96 and '97).

I have been fortunate to encounter many excellent educators and industry people who care and think about the same issues. The debates and new ideas generated in discussions, both live and by email, have allowed me to make the plethora of informed decisions necessary for producing a high-quality textbook. I wish to acknowledge the following people (listed in reverse alphabetical order) with apologies to those whom I have unintentionally left out: Gene Wallingford, Doug Van Weiren, David Teague, Dave Richards, Stuart Reges, Margaret Reek, Ken Reek, Rich Pattis, Linda Northrop, Zung Nguyen, John McCormick, Carolina McCluskey, Mary Lynn Manns, Mike Lutz, David Levine, Jim Heliotis, Peter Grogono, Adele Goldberg, Michael Feldman, Ed Epp, Robert Duvall (not the actor, the guy from Duke), Ward Cunningham, Alistair Cockburn, Mike Clancy, Tim Budd, Barbara Boucher-Owens, Michael Berman, Joe Bergin, Owen Astrachan, and Erzebet Angster. In addition, my thanks go to the following individuals at Franklin, Beedle & Associates: Jim Leisy, Dan Stoops, Stephanie Welch, Susan Skarzynski, Tom Sumner, Sue Page, Cary Crossland, Carrie Widman, Jason Smith, Marc Chambers, and Chris Alarid.

Though too numerous to mention, I also acknowledge the many authors and presenters who have influenced me during my 17-year career in this field.

Reviewers spend countless hours poring over material with critical eyes and useful comments. Because of the high quality of their work, criticisms and recommendations were always considered seriously. I thank the reviewers of both the first edition and of this second edition:

Seth Bergman	*Rowan University*
Michael Berman	*Rowan University*

Tom Bricker	*University of Wisconsin, Madison*
David Teague	*Western Carolina University*
Ed Epp	*University of Portland*
James Murphy	*California State University, Chico*
Rich Pattis	*Carnegie Mellon University*
Jerry Weltman	*Louisiana State University, Baton Rouge*

Reviewers of the second edition:

John Miller	*St. John's University*
Stephen Leach	*Florida State University*
Alva Thompson	*University of South Florida*
Norman Jacobson	*University of California, Irvine*
David Levine	*Gettysburg College*
H. E. Dunsmore	*Purdue University*
Howard Pyron	*University of Missouri at Rolla*
Lee Cornell	*Mankato State University*
Eugene Wallingford	*University of Northern Iowa*
David Teague	*Western Carolina University*
Michael Berman	*Rowan University*
Clayton Lewis	*University of Colorado*
Tim Budd	*Oregon State University*
Jim Miller	*University of Kansas*
Art Farley	*University of Oregon*
Richard Enbody	*Michigan State University*
Van Howbert	*Colorado State University*
Joe Burgin	*Texas Tech University*
Robert Duvall	*Duke University*
Jim Coplien	*Bell Labs*
Dick Weide	*Ohio State University*
Gene Norris	*George Mason University*

Table of Contents