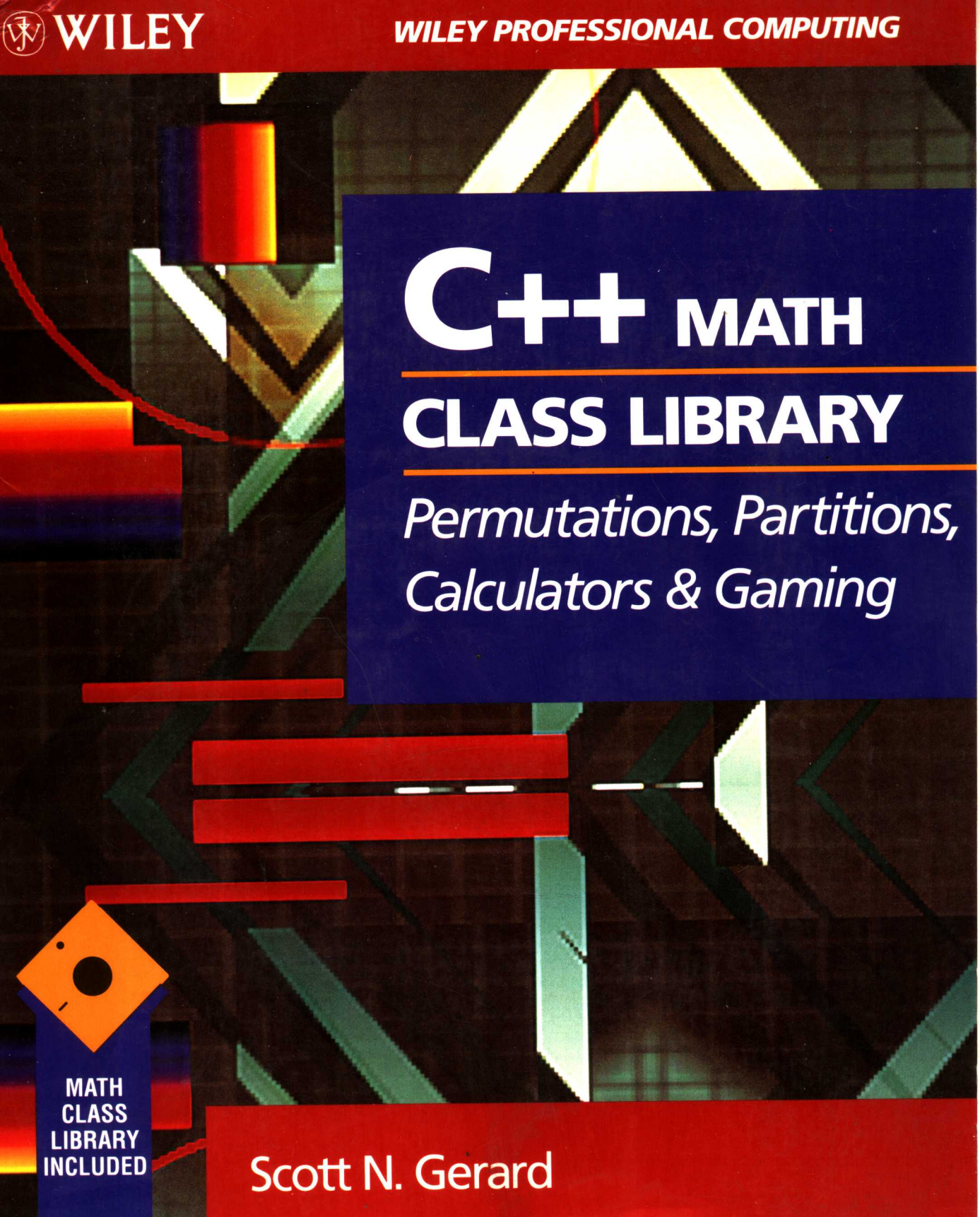
WILEY
WILEY PROFESSIONAL COMPUTING
C++ MATH
CLASS LIBRARY
Permutations, Partitions,
Calculators & Gaming
MATH
CLASS
LIBRARY
INCLUDED
Scott N. Gerard

C++ Math Class Library

Permutations, Partitions, Calculators, and Gaming

Scott N. Gerard

JOHN WILEY & SONS, INC.
New York Chichester Brisbane Toronto Singapore

Associate Publisher: Katherine Schowalter
Editor: Diane Cerra
Managing Editor: Frank Grazioli
Editorial Production & Design: Lachina Publishing Services

This text is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data:

Gerard, Scott N., 1956-
C++ math class library : permutations, partitions, calculators & gaming / by Scott N. Gerard.
p. cm.
Includes bibliographical references and index.
ISBN 0-471-59243-9 (acid-free)
1. C++ (Computer program language) 2. Mathematics—Data processing. I. Title. II. Title: C plus plus math class library.
QA76.73.C153G47 1994
510 ' .285 ' 5133—dc20 93-29732
CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

PREFACE

This book is for programmers who are interested in mathematics and familiar with C++. It is for programmers who are tired of reinventing code that acts "kinda-like" an existing, well-known concept. It is a book for programmers who know that mathematics is full of well-known and useful concepts and who want to tap into these concepts and their power. It is a book for programmers who want to pick up existing code, plug it into their applications, and increase their productivity.

One of the best ways to be more productive is to *stop* writing code and reuse existing programs, particularly for well-defined problems that can be, or have been, solved once and for all. We need libraries of ready-to-run routines.

All of us have written many subroutines in the past. Why not just take all of those routines and tie them up with a pretty bow and call it a library? If only things were that easy. Most subroutines are written with too many assumptions and hooks into their main program. There is a natural tendency to do this. But all those dependencies force us to spend time duplicating their environment or tweaking the code.

Writing general-purpose code is more difficult than writing single-purpose code. Instead of having the exact details of the main program in front of us, we must try to anticipate any and all possible clients. Instead of providing only the function and options required for the specific task at hand, we must provide all relevant operations.

I believe object-oriented (OO) programming makes it easier to write general-purpose code than in non-OO paradigms. Instead of focusing on some hypothetical client and trying to figure out all of its needs, OO focuses on specific objects and asks what operations make sense for that object. OO is not PP (panacea-programming). But although OO has been "overhyped" recently, it does encourage a way of thinking about problems that I believe will be as much a part of future languages as strong data typing has become. Simply put, OO is a good idea and it helps organize programming.

My goals for this book are to

- provide C++ source code for ready-to-run classes,
- leverage the power of mathematical concepts that have stood the test of time,
- give you some ideas where you might apply these concepts in programming applications, and
- teach you some interesting mathematics along the way.

For those of you who like to know the "why" as well as the "what," some proofs are included. But the proofs are clearly marked and can be skipped.

Mid-level Functions

What kinds of routines should be in a programmer's library? A library should not contain programs for entire applications. Complete applications are usually too specific to be heavily reused. And the number of distinct applications is far too large for any reasonable library. Complete applications are just too large for inclusion in a library.

Many books have been published with titles like "A Zillion Little Programs for Your PC." These books are filled with routines to turn on and off the PC speaker, switch video display modes, and so on. These routines fill a certain niche, but are usually not portable to other machines. Primitive functions are needed to write more complex programs, but they do not provide very much function by themselves. They will account for only a small percentage of code in your applications. Therefore, they are too small to greatly increase your productivity because you still have a lot of code to write.

The library routines we're looking for should be not too small and not too big. A library should be made up of medium-sized routines in terms of both size and complexity. Library routines should be complex enough that they can completely take over all processing in one area of an application. And this suggests writing library routines as objects that know how to maintain themselves.

Unusual Classes

In almost every new programming book—regardless of language—the authors present the ever-popular stacks and queues. Stacks and queues are good examples to illustrate the concepts of data hiding, abstraction, and encapsulation. They are small enough that they can be presented without going through a lot of code, and they are truly useful. However, I will assume you already have a number of these books and therefore do not need yet another version.

Instead, I intend to provide other useful data types that are "above" these basic types, that is, data types that are more complex and provide richer function. These routines are portable (or nearly so) to any machine with a C++ compiler. Some of the classes (in particular, the calculator classes) depend on the lists and sets in Borland's class library. If you want to compile my classes on a machine without Borland's classes, you will need to do some reworking.

This book provides a collection of data types that are out of the ordinary, and are intermediate in both size and complexity. There are no stacks or queues, and no sorting routines. All classes are related, in one way or another, to mathematics. This book contains the following classes:

Functions In programming, functions are useful as lookup tables, and for representing finite state machines.

Perm	Permutations have many uses. Besides being a fundamental abstraction of one-to-one and onto functions, they can model card games and other modern puzzles like Rubik's Cube.
Part	Partitions are ideal for representing the concepts of equality and connection.
Polya	This handles unusual types of enumeration problems, like the number of distinct ways to paint the faces of a cube, or the number of distinct bracelets.
Calculator	These classes make it easy to create calculators for your data types.
Region	A region is a mapping between points on a plane, or in a space, and the integers. The region classes support rectangular, triangular, and trapezoidal regions in two dimensions as well as cubical, tetrahedral, pyramidal, and other types of regions in three dimensions.
Xform	These are classes for transforming points including translations, transformations by a group, and general linear transformations.
Hexgrid	These routines manipulate grids of hexagons, which are commonly used in simulation and fantasy games.
Enum IO	These routines read and write enumerations by name.
Hashing	This class combines data into a hash value.
Binomial	In addition to computing binomial coefficients, these routines convert between binary integers and "cogets" (see Chapter 3) in the binomial numbering system.

C++

All the code in this book is in C++ because C++ is the mostly widely used object-oriented language today. This book does *not* cover the basics of the C++ language. There are many good books on C++; for example, see Stroustrup (1991), Lippman (1989), and Meyers (1992).

I will not mount a major defense of the merits of object-oriented programming in C++. If you are reading this book, you probably already agree that C++ is useful, productive, and just plain fun. I will say that I think it is easier to consider all the operations a specific object can reasonably support, than to try and imagine all possible requests from some hypothetical client. This gives me greater confidence that my classes are complete.

Trademarks

Acknowledgments

There are a great many people I want to thank for their help and support of this project. Nancy Hankins, Dave Borrillo, and Craig Orcutt deserve special thanks. They helped

get me started, checked my results, and kept me going. I'd also like to thank Joe and Sue Cahill, Joe Collette, Auther Eberiel, Diane Gerard, Paul Gunsch, Charles and Christian Hankins, Tim Hamel, Jim Herring, Eric Johnson, Steve Knight, Mike Moore, A. Carolyn Neal, Jeff Palm, Curt Rose, Dave and Karen Scudiero, Abolfaz Sirjani, Mike Smith, Francine Stenzel, and Tom Turner.

CONTENTS

FIGURES

PART ONE

Introduction