C++ and Object-Oriented Numeric Computing for Scientists and Engineers

Daoqi Yang

C++ and Object-Oriented Numeric Computing for Scientists and Engineers

Springer

Daoqi Yang
Department of Mathematics
Wayne State University
656 West Kirby Street
Detroit, MI 48202
USA
yang@math.wayne.edu

Library of Congress Cataloging-in-Publication Data

Yang, Daoqi.

C++ and object-oriented numeric computing for scientists and engineers / Daoqi Yang.
p. cm.

Includes bibliographical references and index.

ISBN 978-1-4612-6566-5

ISBN 978-1-4613-0189-9 (eBook)

DOI 10.1007/978-1-4613-0189-9

1. C++ (Computer program language) 2. Object-oriented programming (Computer science) I. Title.

QA76.73.C153 Y35 2000

502'.85'5133—dc21

00-044009

Printed on acid-free paper.

© 2001 Springer Science+Business Media New York

Originally published by Springer-Verlag New York, Inc in 2001

Softcover reprint of the hardcover 1st edition 2001

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher Springer Science+Business Media, LLC, except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

9 8 7 6 5 4 3

ISBN 978-1-4612-6566-5

SPIN 10946183

To Xiujuan, Joy, and Forrest

Preface

This book is intended to be an easy, concise, but rather complete, introduction to the ISO/ANSI C++ programming language with special emphasis on object-oriented numeric computation for students and professionals in science and engineering. The description of the language is platform-independent. Thus it applies to different operating systems such as UNIX, Linux, MacOS, Windows, and DOS, as long as a standard C++ compiler is equipped. The prerequisite of this book is elementary knowledge of calculus and linear algebra. However, this prerequisite is hardly necessary if this book is going to be used as a textbook for teaching C++ and all the sections on numeric methods are skipped. Programming experience in another language such as FORTRAN, C, Ada, Pascal, Maple, or Matlab will certainly help, but is not presumed.

All C++ features are introduced in an easy way through concepts such as functions, complex numbers, vectors, matrices, and integrals, which are familiar to every student and professional in science and engineering. In the final chapter, advanced features that are not found in FORTRAN, C, Ada, or Matlab, are illustrated in the context of iterative algorithms for linear systems such as the preconditioned conjugate gradient (CG) method and generalized minimum residual (GMRES) method. Knowledge of CG, GMRES, and preconditioning techniques is not presumed and they are explained in detail at the algorithmic level. Matrices can be stored in full (all entries are stored), band (entries outside a band along the main diagonal are zero and not stored to save memory), and sparse (only nonzero entries are stored to save memory) formats and exactly one definition for CG or GMRES is needed that is good for all three matrix storage formats. This is

in contrast to a procedural language such as FORTRAN, C, Ada, or Matlab, where three definitions of CG and GMRES may have to be given for the three matrix formats. This is one of the salient features of object-oriented programming called *inheritance*. The CG and GMRES methods are defined for a base *class* and can be inherited by classes for full, band, and sparse matrices. If one later decides to add, for example, a symmetric (full, band, or sparse) storage format (only half of the entries need be stored for a symmetric matrix to save memory), the code of CG and GMRES methods can be reused for it without any change or recompilation.

Another notable feature is generic programming through *templates*, which enables one to define a function that may take arguments of different data types at different invocations. For example, the same definition of CG and GMRES can deal with matrices with entries in single, double, and extended double (*long double*) precisions and complex numbers with different types for real and imaginary parts. Again, using C or FORTRAN would require different versions of the same function to handle different data types; when the implementation of the function is going to be changed later for robustness or efficiency, then every version has to be changed, which is tedious and error-prone. The *operator overloading* feature of C++ enables one to add and multiply matrices and vectors using + and * in the same way as adding and multiplying numbers. The meaning of the operators is user-defined and thus it provides more flexibility than Matlab. For example, the operators + and * can be used to add and multiply full, band, and sparse matrices. These and other features of C++ such as information hiding, encapsulation, polymorphism, error handling, and standard libraries are explained in detail in the text. With increasingly complicated numeric algorithms, many sophisticated data structures can be relatively easily implemented in C++, rather than in FORTRAN or C; this is a very important feature of C++. The C++ compiler also checks for more type errors than FORTRAN, C, Ada, Pascal, and many other languages do, which makes C++ safer to use.

However, there are trade-offs. For example, *templates* impose compile-time overhead and *inheritance* (with dynamic type binding) may impose run-time overhead. These features could slow down a program at compile- and run-times. A good C++ compiler can minimize these overheads and a programmer can still enjoy programming in C++ without suffering noticeable compile- and run-time slowdowns but with spending possibly much less time in code development. This is evidenced by the popularity of C++ in industry for database, graphics, and telecommunication applications, where people also care dearly about the speed of a language. The fact is that C++ is getting faster as people are spending more time optimizing the compiler. Some performance comparisons on finite element analysis have shown that C++ is comparable to FORTRAN 77 and C in terms of speed. On the other hand, C++ has the potential of outperforming FORTRAN 77 and C for CPU-intensive numeric computations, due to its built-in arithmetic compound operators, template mechanisms that can, for example,

avoid the overhead of passing functions as arguments to other functions, and high performance libraries (like *valarray*), which can be optimized on different computer architectures. This potential has become a reality in many test examples.

This book consists of three parts. The first part (Chapters 1 to 4) is an introduction to basic features of C++, which have equivalents in FORTRAN 90 or C. When a person knows only this part, he can do whatever a FORTRAN 90 or C programmer can do. A brief introduction is also made to *Makefile*, debuggers, making a library, and how to time and profile a program. The second part (Chapters 5 to 9) is on object-oriented and generic programming features of C++, which can not be found in FORTRAN 90 or C. Many techniques for writing efficient C++ programs are also included; see §6.5, §7.2, §7.6, §7.2.4, §7.7, and §8.6. It is this part that causes many people to think that C++ is “too complicated.” To make it easy to understand for a typical science and engineering student or professional, I make use of basic concepts and operations for complex numbers, vectors, matrices, and integrals. This should be readily acceptable by a person with elementary knowledge of calculus and matrix algebra. The third part (Chapters 10 and 11) provides a description of C++ standard libraries on containers (such as linked list, set, vector, map, stack, and queue) and algorithms (such as sorting a sequence of elements and searching an element from a sequence according to different comparison criteria), and an introduction to a user-defined numeric linear algebra library (downloadable from my Web page) that contains functions for preconditioned conjugate gradient and generalized minimum residual methods for real and complex matrices stored in full, band, and sparse formats. Gauss elimination with and without pivoting is also included for full and band matrices. This enhances the reader’s understanding of Parts 1 and 2. Furthermore, it can save the reader a great deal of time if she has to write her own basic numeric linear algebra library, which is used on a daily basis by many scientists and engineers. Great care has been taken so that features that are more easily understood and that are more often used in numeric computing are presented first. Some examples in the book are taken from or can be used in numeric computing libraries, while others are made up only to easily illustrate features of the C++ language. A Web page (<http://www.math.wayne.edu/~yang/book.htm>) is devoted to the book on information such as errata and downloading programs in the book.

On the numeric side, discussions and programs are included for the following numeric methods in various chapters: polynomial evaluation (§3.12), numeric integration techniques (§3.13, §5.1, and §7.7), vector and matrix arithmetic (§6.3 and §7.1), iterative algorithms for solving nonlinear equations (§4.7), polynomial interpolation (§7.8), iterative and direct algorithms for solving systems of linear equations in real and complex domains (§6.6, §11.3, and §11.4), Euler and Runge-Kutta methods for solving ordinary differential equations (§5.9), and a finite difference method for solving par-

tial differential equations (§11.5) with real and complex coefficients. The coverage of numeric algorithms is far from being complete. It is intended to provide a basic understanding of numeric methods and to see how C++ can be applied to program these methods efficiently and elegantly. Most of them are covered at the end of a chapter. People with an interest in learning how to program numeric methods may read them carefully, while readers who just want to learn C++ may wish to skip them or read them briefly.

C++ is not a perfect language, but it seems to have all the features and standard libraries of which a programmer can dream. My experience is that it is much easier to program than FORTRAN and C, because FORTRAN and C give too many run-time errors and have too few features and standard libraries. Many such run-time errors are hard to debug but can easily be caught by a C++ compiler.

After all, C++ is just a set of notation to most people and a person does not have to know all the features of C++ in order to write good and useful programs. Enjoy!

How to Use This Book:

This book can be used as a textbook or for self-study in a variety of ways.

1. The primary intent of this book is to teach C++ and numeric computing at the same time, for students and professionals in science and engineering. C++ is first introduced and then applied to code numeric algorithms.
2. It can also be used for people who just want to learn basic and advanced features of C++. In this case, sections on numeric computing can be omitted, and knowledge of calculus and linear algebra is not quite necessary. However, in some sections, the reader should know what a vector and a matrix are, and basic operations on vectors and matrices, such as vector addition, scalar-vector multiplication, and matrix-vector multiplication.
3. This book can be used as a reference for people who are learning numeric methods, since C++ programs of many numeric methods are included. Students who learn numeric methods for the first time often have difficulty programming the methods. The C++ code in the book should help them get started and have a deeper understanding of the numeric methods.
4. For experienced C++ programmers, this book may be used as a reference. It covers essentially all features of the ISO/ANSI C++ programming language and libraries. It also contains techniques for writing efficient C++ programs. For example, standard operator overloading for vector operations, as introduced in most C++ books, may be

a few times slower than using C or FORTRAN style programming. Techniques are given in the book so that the C++ code is no slower than its corresponding C or FORTRAN style code. Examples are also given to replace certain virtual functions by static polymorphism to improve run-time efficiency. Other advanced techniques include expression templates and template metaprograms.

Acknowledgments:

This book has been used as the textbook for a one-semester undergraduate course on C++ and numeric computing at Wayne State University. The author would like to thank his students for valuable suggestions and discussions. Thanks also go to the anonymous referees and editors whose careful review and suggestions have improved the quality of the book. Readers' comments and suggestions are welcome and can be sent to the author via email (dyang@na-net.ornl.gov).

Daoqi Yang
Wayne State University
Detroit, Michigan

June 28, 2000

Contents

Preface	vii
1 Basic Types	1
1.1 A Sample Program	1
1.2 Types and Declarations	6
1.3 Basic Types	8
1.3.1 Integer Types	8
1.3.2 Character Types	10
1.3.3 Floating Point Types	11
1.3.4 The Boolean Type	14
1.3.5 The Void Type	14
1.4 Numeric Limits	15
1.5 Identifiers and Keywords	19
1.5.1 Identifiers	19
1.5.2 Keywords	20
1.6 Exercises	21
2 Expressions and Statements	25
2.1 Scopes and Storage Classifications	25
2.1.1 Local and Global Variables	26
2.1.2 External and Register Variables	27
2.2 Expressions	28
2.2.1 Arithmetic Expressions	28
2.2.2 Relational Expressions	29

2.2.3	Logical Expressions	30
2.2.4	Bitwise Expressions	31
2.2.5	Comma Expressions	33
2.3	Statements	33
2.3.1	Declarations and Initializations	34
2.3.2	Assignments	35
2.3.3	Compound Assignments	35
2.3.4	Increments and Decrements	36
2.3.5	Compound Statements	36
2.3.6	Conditional Statements	37
2.3.7	Iteration Statements	41
2.4	Fibonacci Numbers	44
2.5	Exercises	47
3	Derived Types	51
3.1	Constants and Macros	51
3.2	Enumerations	52
3.3	Arrays	54
3.4	Structures	57
3.5	Unions and Bit Fields	57
3.6	Pointers	60
3.6.1	Pointer Arithmetic	64
3.6.2	Multiple Pointers	64
3.6.3	Offsetting Pointers	67
3.6.4	Constant Pointers	68
3.6.5	Void and Null Pointers	70
3.6.6	Pointers to Structures	70
3.6.7	Pointers to Char	72
3.6.8	Pointers and Arrays	72
3.7	References	72
3.8	Functions	73
3.8.1	Function Declarations and Definitions	73
3.8.2	Function Overloading	74
3.8.3	Argument Passing	75
3.8.4	Return Values	79
3.8.5	Recursive Functions	81
3.8.6	Inline Functions	81
3.8.7	Default Arguments	82
3.8.8	Function Types	83
3.8.9	Static Local Variables	84
3.8.10	The Function <i>main</i>	85
3.9	Program Execution	86
3.10	Operator Summary and Precedence	88
3.11	Standard Library on Mathematical Functions	92
3.12	Polynomial Evaluation	94

3.13	Trapezoidal and Simpson's Rules	98
3.14	Exercises	101
4	Namespaces and Files	113
4.1	Namespaces	113
4.1.1	Using Declarations and Directives	117
4.1.2	Multiple Interfaces	119
4.1.3	Namespace Alias	120
4.1.4	Unnamed Namespaces	121
4.1.5	Name Lookup	121
4.2	Include Files	122
4.2.1	Include Files for Standard Libraries	122
4.2.2	User's Own Include Files	124
4.2.3	Conditional Include Directives	126
4.2.4	File Inclusion	128
4.3	Source Files and Linkages	129
4.3.1	Separate Compilation	129
4.3.2	External and Internal Linkages	129
4.3.3	Linkage to Other Languages	133
4.4	Some Useful Tools	134
4.4.1	How to Time a Program	134
4.4.2	Compilation Options and Debuggers	136
4.4.3	Creating a Library	138
4.4.4	Makefile	139
4.5	Standard Library on Strings	142
4.5.1	Declarations and Initializations	142
4.5.2	Operations	143
4.5.3	C-Style Strings	144
4.5.4	Input and Output	144
4.5.5	C Library on Strings	145
4.6	Standard Library on Streams	146
4.6.1	Formatted Integer Output	146
4.6.2	Formatted Floating Point Output	148
4.6.3	Output Width	149
4.6.4	Input and Output Files	150
4.6.5	Input and Output of Characters	153
4.6.6	String Streams	157
4.7	Iterative Methods for Nonlinear Equations	158
4.7.1	Bisection Method	159
4.7.2	Newton's Method	165
4.8	Exercises	167
5	Classes	173
5.1	Class Declarations and Definitions	173
5.2	Copy Constructors and Copy Assignments	180

5.3	Friends	184
5.4	Static Members	185
5.5	Constant and Mutable Members	187
5.6	Class Objects as Members	189
5.7	Array of Classes	191
5.8	Pointers to Members	193
5.9	Numeric Methods for Ordinary Differential Equations	194
5.10	Exercises	199
6	Operator Overloading	203
6.1	Complex Numbers	204
6.1.1	Initialization	206
6.1.2	Default Copy Construction and Assignment	206
6.1.3	Conversions and Mixed-Mode Operations	207
6.2	Operator Functions	207
6.3	Vectors and Matrices	211
6.4	Explicit and Implicit Conversions	220
6.5	Efficiency and Operator Overloading	223
6.6	Conjugate Gradient Algorithm	226
6.7	Exercises	229
7	Templates	231
7.1	Class Templates	231
7.1.1	Member and Friend Definitions	233
7.1.2	Template Instantiation	234
7.1.3	Template Parameters	235
7.1.4	Type Equivalence	235
7.1.5	User-Defined Specializations	235
7.1.6	Order of Specializations	239
7.2	Function Templates	240
7.2.1	Function Template Parameters	241
7.2.2	Function Template Overloading	242
7.2.3	Specializations	243
7.2.4	Class Templates as Function Template Parameters	245
7.2.5	Member Function Templates	246
7.2.6	Friend Function Templates	247
7.3	Template Source Code Organization	248
7.4	Standard Library on Complex Numbers	250
7.5	Standard Library on <i>valarrays</i>	251
7.5.1	The Type <i>valarray</i>	252
7.5.2	Slice Arrays	253
7.5.3	Generalized Slice Arrays	256
7.5.4	Mask Arrays and Indirect Arrays	257
7.6	Standard Library on Numeric Algorithms	258
7.6.1	Accumulate	259

7.6.2	Inner Products	262
7.6.3	Partial Sums	263
7.6.4	Adjacent Differences	263
7.7	Efficient Techniques for Numeric Integration	264
7.7.1	Function Object Approach	264
7.7.2	Function Pointer as Template Parameter	265
7.7.3	Using Dot Products and Expression Templates	266
7.7.4	Using Dot Products and Template Metaprograms	270
7.8	Polynomial Interpolation	273
7.8.1	Lagrange Form	273
7.8.2	Newton Form	275
7.9	Exercises	279
8	Class Inheritance	283
8.1	Derived Classes	283
8.1.1	Member Functions	286
8.1.2	Constructors and Destructors	286
8.1.3	Copying	287
8.1.4	Class Hierarchy	287
8.1.5	Virtual Functions	288
8.1.6	Virtual Destructors	289
8.2	Abstract Classes	291
8.3	Access Control	296
8.3.1	Access to Members	296
8.3.2	Access to Base Classes	297
8.4	Multiple Inheritance	300
8.4.1	Ambiguity Resolution	301
8.4.2	Replicated Base Classes	302
8.4.3	Virtual Base Classes	303
8.4.4	Access Control in Multiple Inheritance	304
8.5	Run-Time Type Information	305
8.5.1	The <i>dynamic_cast</i> Mechanism	306
8.5.2	The <i>typeid</i> Mechanism	308
8.5.3	Run-Time Overhead	309
8.6	Replacing Virtual Functions by Static Polymorphism	309
8.7	Exercises	316
9	Exception Handling	319
9.1	Throw and Catch	319
9.2	Deriving Exceptions	323
9.3	Catching Exceptions	325
9.3.1	Re-throw	326
9.3.2	Catch All Exceptions	326
9.3.3	Order of Handlers	327
9.4	Specifying Exceptions in Functions	327

9.5	Standard Exceptions	329
9.6	Exercises	331
10	Standard Libraries on Containers and Algorithms	333
10.1	Standard Containers	333
10.1.1	Vector	334
10.1.2	List	341
10.1.3	Map and Set	344
10.1.4	Stack and Queue	347
10.2	Standard Algorithms	348
10.2.1	Sorting, Copying, and Replacing Algorithms	348
10.2.2	Searching and Traversing Algorithms	355
10.2.3	Set, Permutation, and Heap Algorithms	360
10.3	Standard Function Objects and Adaptors	365
10.3.1	Arithmetic Function Objects	365
10.3.2	Relational Function Objects	366
10.3.3	Logical Function Objects	366
10.3.4	Standard Adaptors	367
10.4	Exercises	368
11	Linear System Solvers	371
11.1	Matrix Storage Formats	372
11.1.1	Full Matrices	372
11.1.2	Band Matrices	372
11.1.3	Sparse Matrices	374
11.2	A Class Hierarchy for Matrices	375
11.3	Iterative Algorithms	385
11.3.1	Conjugate Gradient Method	385
11.3.2	Generalized Minimum Residual Method	390
11.3.3	Preconditioning Techniques	398
11.4	Gauss Elimination	401
11.4.1	LU Decomposition	401
11.4.2	Gauss Elimination Without Pivoting	406
11.4.3	Gauss Elimination with Pivoting	408
11.5	Finite Difference Method for Partial Differential Equations	414
11.6	Exercises	424
	References	427
	Index	430