
Undergraduate Topics in Computer Science

Series editor

Ian Mackie

Advisory Board

Samson Abramsky, University of Oxford, Oxford, UK

Chris Hankin, Imperial College London, London, UK

Mike Hinchey, University of Limerick, Limerick, Ireland

Dexter C. Kozen, Cornell University, Ithaca, USA

Andrew Pitts, University of Cambridge, Cambridge, UK

Hanne Riis Nielson, Technical University of Denmark, Kongens Lyngby, Denmark

Steven S. Skiena, Stony Brook University, Stony Brook, USA

Iain Stewart, University of Durham, Durham, UK

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at <http://www.springer.com/series/7592>

Joe Pitt-Francis · Jonathan Whiteley

Guide to Scientific Computing in C++

Second Edition

 Springer

Joe Pitt-Francis
University of Oxford
Oxford
UK

Jonathan Whiteley
University of Oxford
Oxford
UK

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-73131-5 ISBN 978-3-319-73132-2 (eBook)
<https://doi.org/10.1007/978-3-319-73132-2>

Library of Congress Control Number: 2017962059

1st edition: © Springer-Verlag London Limited 2012

2nd edition: © Springer International Publishing AG, part of Springer Nature 2017

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG part of Springer Nature

The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface to the Second Edition

The principle changes in this updated edition are additional material on software testing and on some of the new features introduced in the C++11 standard. When introducing this additional material, we have followed the same philosophy as when writing the first edition of this book. That is, we focus on a concise discussion of the key features that are most useful to the novice and intermediate programmer in the field of scientific computing. We have found this an effective approach when teaching this course to graduate students—once the basics have been mastered, students then have the confidence to find out about less well-used features themselves when they are needed.

This second edition would not be as complete—or as enjoyable to update—without discussions with colleagues and other readers of the first edition, including those previously unknown to us who were kind enough to provide constructive feedback. We would like to express our gratitude to all who contributed in this way or offered their encouragement, and to the staff at Springer for inviting us to update the first edition.

Finally, we would both again like to thank our families for their love and support.

Oxford, UK
October 2017

Joe Pitt-Francis
Jonathan Whiteley

Preface to the First Edition

Many books have been written on the C++ programming language, varying across a spectrum from the very practical to the very theoretical. This book certainly lies at the practical end of this spectrum and has a particular focus for the practical treatment of this language: scientific computing.

Traditionally, Fortran and MATLAB^{®1} have been the languages of choice for scientific computing applications. The recent development of complex mathematical models—in fields as diverse as biology, finance and materials science, to name but a few—has driven a need for software packages that allow computational simulations based on these models. The complexity of the underlying models, together with the need to exchange code between co-workers, has motivated programmers to develop object-oriented code (often written in C++) for these simulation packages. The computational demands of these simulations may require software to be written for parallel computing facilities, typically using the Message Passing Interface (MPI). The need to train programmers in the skills to program applications such as these led to the development of a graduate-level course *C++ for Scientific Computing*, taught by the authors of this book, at the University of Oxford.

This book provides a guide to C++ programming in scientific computing. In contrast to many other books on C++, features of the language are demonstrated mainly using examples drawn from scientific computing. Object orientation is first mentioned in Chap. 1 where we briefly describe what this phrase—and other related terms such as inheritance—means, before postponing any further discussion of object orientation or related topics until Chap. 6. In the intervening chapters until object orientation reappears, we present what is best described as “procedural programming in C++”, covering variables, flow of control, input and output, pointers (including dynamic allocation of memory), functions and reference variables. Armed with this grounding in C++, we then introduce classes in Chaps. 6 and 7. In these two chapters, where the main features of object orientation are showcased, we

¹MATLAB is a registered trademark of The MathWorks, Inc.

initially, for the sake of clarity, abandon our principle of using examples drawn from scientific computing. Once the topics have been presented however, we resume our strategy of demonstrating concepts through scientific computing examples. More advanced C++ features such as templates and exceptions are introduced in Chaps. 8 and 9. Having introduced the features of C++ required for scientific computing, the remainder of the book focuses on the application of these features. In Chap. 10, we begin to develop a collection of classes for linear algebra calculations: these classes are then developed further in the exercises at the end of this chapter. Chapter 11 presents an introduction to parallel computing using MPI. Finally, in Chap. 12, we discuss how an object-oriented library for solving second-order differential equations may be constructed. The importance of a clear programming style to minimise the introduction of errors into code is stressed throughout the book.

This book is aimed at programmers of all levels of expertise who wish to write scientific computing programs in C++. Experience with a computer to the level where files can be stored and edited is expected. A basic knowledge of mathematics, such as operations between vectors and matrices, and the Newton–Raphson method for finding the roots of nonlinear equations would be an advantage.

The material presented here has been enhanced significantly by discussions about C++ with colleagues, too numerous to list here, in the Department of Computer Science at the University of Oxford. A special mention must, however, be made of the Chaste² programming team: particular gratitude should be expressed to Jonathan Cooper for readily sharing with us his impressively wide and deep knowledge of the C++ language. Other members of the team who have significantly helped clarify our thoughts on the C++ language are Miguel Bernabeu, James Osborne, Pras Pathmanathan and James Southern. We should also thank students from both the M.Sc. in Mathematical Modelling and Scientific Computing and the Doctoral Training Centres at the University of Oxford for unwittingly aiding our understanding of the language through asking pertinent questions.

Finally, it is always important to remember—especially when debugging a particularly tiresome code—that there is far more to life than C++ programming for scientific computing. We would both like to thank our families for their love and support, especially during the writing of this book.

Oxford
October 2011

Joe Pitt-Francis
Jonathan Whiteley

²The Cancer, Heart And Soft Tissue Environment (Chaste) is an object-oriented package, written in C++, for simulations in the field of biology. More details on this package may be found at <https://www.cs.ox.ac.uk/chaste/>.

Contents

1	Getting Started	1
1.1	A Brief Introduction to C++	1
1.1.1	C++ is “Object-Oriented”	2
1.1.2	Why You Should Write Scientific Programs in C++	3
1.1.3	Why You Should Not Write Scientific Programs in C++	4
1.1.4	Scope of This Book	5
1.2	A First C++ Program	5
1.3	Compiling a C++ Program	6
1.3.1	Integrated Development Environments	7
1.3.2	Compiling at the Command Line	8
1.3.3	Compiler Flags	9
1.4	Variables	10
1.4.1	Basic Numerical Variables	10
1.4.2	Other Numerical Variables	12
1.4.3	Mathematical Operations on Numerical Variables	14
1.4.4	Division of Integers	16
1.4.5	Arrays	17
1.4.6	ASCII Characters	18
1.4.7	Boolean Variables	19
1.4.8	Strings	19
1.5	Simple Input and Output	20
1.5.1	Basic Console Output	20
1.5.2	Keyboard Input	21
1.6	The <code>assert</code> Statement	22
1.7	Tips: Debugging Code	24
1.8	Exercises	25
2	Flow of Control	27
2.1	The <code>if</code> Statement	28
2.1.1	A Single <code>if</code> Statement	28
2.1.2	Example: Code for a Single <code>if</code> Statement	29

2.1.3	if-else Statements	29
2.1.4	Multiple if Statements	30
2.1.5	Nested if Statements	30
2.1.6	Boolean Variables	31
2.2	Logical and Relational Operators	31
2.3	The while Statement	33
2.4	Loops Using the for Statement	35
2.4.1	Example: Calculating the Scalar Product of Two Vectors	36
2.5	The switch Statement	37
2.6	Tips: Loops and Branches	38
2.6.1	Tip 1: A Common Novice Coding Error	38
2.6.2	Tip 2: Counting from Zero	38
2.6.3	Tip 3: Equality Versus Assignment	39
2.6.4	Tip 4: Never Ending while Loops	41
2.6.5	Tip 5: Comparing Two Floating Point Numbers	41
2.7	Exercises	42
3	File Input and Output	47
3.1	Redirecting Console Output to File	47
3.2	Writing to File	48
3.2.1	Setting the Precision of the Output	51
3.3	Reading from File	51
3.4	Checking Input and Output are Successful	53
3.5	Reading from the Command Line	54
3.6	Tips: Controlling Output Format	55
3.7	Exercises	56
4	Pointers	59
4.1	Pointers and the Computer's Memory	59
4.1.1	Addresses	59
4.1.2	Pointer Variables	60
4.1.3	Example Use of Pointers	61
4.1.4	Warnings on the Use of Pointers	61
4.2	Dynamic Allocation of Memory for Arrays	62
4.2.1	Vectors	63
4.2.2	Matrices	64
4.2.3	Irregularly Sized Matrices	65
4.3	Tips: Pointers	66
4.3.1	Tip 1: Pointer Aliasing	66
4.3.2	Tip 2: Safe Dynamic Allocation	67
4.3.3	Tip 3: Every new Has a delete	68

4.4	Modern C++ Memory Management	69
4.4.1	The <code>unique_ptr</code> Smart Pointer	69
4.4.2	The <code>shared_ptr</code> Smart Pointer	71
4.5	Exercises	72
5	Blocks, Functions and Reference Variables	75
5.1	Blocks	75
5.2	Functions	77
5.2.1	Simple Functions	77
5.2.2	Returning Pointer Variables from a Function	79
5.2.3	Use of Pointers as Function Arguments	80
5.2.4	Sending Arrays to Functions	82
5.2.5	Example: A Function to Calculate the Scalar Product of Two Vectors	84
5.3	Reference Variables	85
5.4	Default Values for Function Arguments	86
5.5	Function Overloading	87
5.6	Declaring Functions Without Prototypes	89
5.7	Function Pointers	89
5.8	Recursive Functions	92
5.9	Modules	93
5.10	Tips: Code Documentation	94
5.11	Exercises	96
6	An Introduction to Classes	99
6.1	The <i>Raison d'Être</i> for Classes	99
6.1.1	Problems That May Arise When Using Modules	100
6.1.2	Abstraction, Encapsulation and Modularity Properties of Classes	100
6.2	A First Example Simple Class: A Class of Books	101
6.2.1	Basic Features of Classes	101
6.2.2	Header Files	103
6.2.3	Setting and Accessing Variables	104
6.2.4	Compiling Multiple Files	107
6.2.5	Access Privileges	109
6.2.6	Including Function Implementations in Header Files	110
6.2.7	Constructors and Destructors	110
6.2.8	Pointers to Classes	115
6.3	The friend Keyword	116
6.4	A Second Example Class: A Class of Complex Numbers	117
6.4.1	Operator Overloading	118
6.4.2	The Class of Complex Numbers	119

6.5	Some Additional Remarks on Operator Overloading	125
6.6	Tips: Coding to a Standard	125
6.7	Exercises	127
7	Inheritance and Derived Classes	129
7.1	Inheritance, Extensibility and Polymorphism	129
7.2	Example: A Class of E-books Derived from a Class of Books	130
7.3	Access Privileges for Derived Classes	133
7.4	Classes Derived from Derived Classes	134
7.5	Run-Time Polymorphism	134
7.6	The Abstract Class Pattern	137
7.7	Tips: Using a Debugger	138
7.8	Exercises	139
8	Templates	145
8.1	Templates to Control Dimensions and Verify Sizes	145
8.2	Templates for Polymorphism	147
8.3	A Brief Survey of the Standard Template Library	148
	8.3.1 Vectors	148
	8.3.2 Sets	151
8.4	A Survey of Some New Functionality in Modern C++	153
	8.4.1 The <code>auto</code> Type	154
	8.4.2 Some Useful Container Types with Unified Functionality	155
	8.4.3 Range-based <code>for</code> Loops	157
	8.4.4 Mapping Lambda Functions	158
8.5	Tips: Template Compilation	159
8.6	Exercises	160
9	Errors, Exceptions and Testing	163
9.1	Preconditions	164
	9.1.1 Example: Two Implementations of a Graphics Function	164
9.2	Three Levels of Errors	166
9.3	Introducing the Exception	167
9.4	Using Exceptions	168
9.5	Testing Software	169
	9.5.1 Unit Testing	170
	9.5.2 Extending Software	171
	9.5.3 Black Box Testing	172
	9.5.4 White Box Testing	176
	9.5.5 Test Driven Development	177
9.6	Tips: Writing Appropriate Tests	178
9.7	Exercises	179

10	Developing Classes for Linear Algebra Calculations	183
10.1	Requirements of the Linear Algebra Classes	183
10.2	Constructors and Destructors	188
10.2.1	The Default Constructor	188
10.2.2	The Copy Constructor	188
10.2.3	A Specialised Constructor	189
10.2.4	Destructor	189
10.3	Accessing Private Class Members	189
10.3.1	Accessing the Size of a Vector	190
10.3.2	Overloading the Square Bracket Operator	190
10.3.3	Read-Only Access to Vector Entries	190
10.3.4	Overloading the Round Bracket Operator	190
10.4	Operator Overloading for Vector Operations	190
10.4.1	The Assignment Operator	191
10.4.2	Unary Operators	191
10.4.3	Binary Operators	191
10.5	Functions	191
10.5.1	Members Versus Friends	191
10.6	Tips: Memory Debugging Tools	192
10.7	Exercises	193
11	An Introduction to Parallel Programming Using MPI	197
11.1	Distributed Memory Architectures	197
11.2	Installing MPI	199
11.3	A First Program Using MPI	199
11.3.1	Essential MPI Functions	201
11.3.2	Compiling and Running MPI Code	201
11.4	Basic MPI Communication	203
11.4.1	Point-to-Point Communication	203
11.4.2	Collective Communication	206
11.5	Example MPI Applications	213
11.5.1	Summation of Series	213
11.5.2	Parallel Linear Algebra	215
11.6	Tips: Debugging a Parallel Program	218
11.6.1	Tip 1: Make an Abstract Program	219
11.6.2	Tip 2: Datatype Mismatch	219
11.6.3	Tip 3: Intermittent Deadlock	220
11.6.4	Tip 4: Almost Collective Communication	220
11.7	Exercises	221
12	Designing Object-Oriented Numerical Libraries	225
12.1	Developing the Library for Ordinary Differential Equations	226
12.1.1	Model Problems	226
12.1.2	Finite Difference Approximation to Derivatives	227

12.1.3	Application of Finite Difference Methods to Boundary Value Problems	229
12.1.4	Concluding Remarks on Boundary Value Problems in One Dimension	231
12.2	Designing a Library for Solving Boundary Value Problems . . .	232
12.2.1	The Class SecondOrderOde	233
12.2.2	The Class BoundaryConditions	234
12.2.3	The Class FiniteDifferenceGrid	235
12.2.4	The Class BvpOde	236
12.2.5	Using the Class BvpOde	237
12.3	Extending the Library to Two Dimensions	238
12.3.1	Model Problem for Two Dimensions	239
12.3.2	Finite Difference Methods for Boundary Value Problems in Two Dimensions	239
12.3.3	Setting Up the Linear System for the Model Problem	241
12.3.4	Developing the Classes Required	242
12.4	Tips: Using Well-Written Libraries	243
12.5	Exercises	243
	Appendix A: Linear Algebra	245
	Appendix B: Other Programming Constructs You Might Meet	257
	Appendix C: Solutions to Exercises	263
	Further Reading	281
	Index	283