

---

# **Undergraduate Topics in Computer Science**

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.

For further volumes:

<http://www.springer.com/series/7592>

---

Joe Pitt-Francis • Jonathan Whiteley

# Guide to Scientific Computing in C++

 Springer

Dr. Joe Pitt-Francis  
Department of Computer Science  
University of Oxford  
Oxford, UK

Dr. Jonathan Whiteley  
Department of Computer Science  
University of Oxford  
Oxford, UK

*Series editor*  
Ian Mackie

*Advisory board*

Samson Abramsky, University of Oxford, Oxford, UK  
Karin Breitman, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil  
Chris Hankin, Imperial College London, London, UK  
Dexter Kozen, Cornell University, Ithaca, USA  
Andrew Pitts, University of Cambridge, Cambridge, UK  
Hanne Riis Nielson, Technical University of Denmark, Kongens Lyngby, Denmark  
Steven Skiena, Stony Brook University, Stony Brook, USA  
Iain Stewart, University of Durham, Durham, UK

ISSN 1863-7310 Undergraduate Topics in Computer Science  
ISBN 978-1-4471-2735-2 e-ISBN 978-1-4471-2736-9  
DOI 10.1007/978-1-4471-2736-9  
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data  
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2012931858

© Springer-Verlag London Limited 2012

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licenses issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

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

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

---

## Preface

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<sup>1</sup> 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 coworkers, 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—mean, 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 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

---

<sup>1</sup>MATLAB is a registered trademark of The MathWorks, Inc.

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 Chaste<sup>2</sup> 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, UK

Joe Pitt-Francis  
Jonathan Whiteley

---

<sup>2</sup>The 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 <http://www.cs.ox.ac.uk/chaste/>.

---

# Contents

<b>1</b>	<b>Getting Started</b>	1
1.1	A Brief Introduction to C++	1
1.1.1	C++ is “Object-Oriented”	1
1.1.2	Why You Should Write Scientific Programs in C++	2
1.1.3	Why You Should Not Write Scientific Programs in C++	4
1.1.4	Scope of This Book	4
1.2	A First C++ Program	5
1.3	Compiling a C++ Program	6
1.3.1	Integrated Development Environments	6
1.3.2	Compiling at the Command Line	7
1.3.3	Compiler Flags	8
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	13
1.4.4	Division of Integers	15
1.4.5	Arrays	16
1.4.6	ASCII Characters	17
1.4.7	Boolean Variables	17
1.4.8	Strings	18
1.5	Simple Input and Output	19
1.5.1	Basic Console Output	19
1.5.2	Keyboard Input	20
1.6	The <code>assert</code> Statement	21
1.7	Tips: Debugging Code	22
1.8	Exercises	23
<b>2</b>	<b>Flow of Control</b>	25
2.1	The <code>if</code> Statement	25
2.1.1	A Single <code>if</code> Statement	26
2.1.2	Example: Code for a Single <code>if</code> Statement	27
2.1.3	<code>if-else</code> Statements	27
2.1.4	Multiple <code>if</code> Statements	27
2.1.5	Nested <code>if</code> Statements	28

2.1.6	Boolean Variables . . . . .	28
2.2	Logical and Relational Operators . . . . .	29
2.3	The <code>while</code> Statement . . . . .	30
2.4	Loops Using the <code>for</code> Statement . . . . .	32
2.4.1	Example: Calculating the Scalar Product of Two Vectors . . . . .	34
2.5	The <code>switch</code> Statement . . . . .	34
2.6	Tips: Loops and Branches . . . . .	35
2.6.1	Tip 1: A Common Novice Coding Error . . . . .	35
2.6.2	Tip 2: Counting from Zero . . . . .	36
2.6.3	Tip 3: Equality Versus Assignment . . . . .	37
2.6.4	Tip 4: Never Ending <code>while</code> Loops . . . . .	38
2.6.5	Tip 5: Comparing Two Floating Point Numbers . . . . .	39
2.7	Exercises . . . . .	39
<b>3</b>	<b>File Input and Output . . . . .</b>	<b>43</b>
3.1	Redirecting Console Output to File . . . . .	43
3.2	Writing to File . . . . .	44
3.2.1	Setting the Precision of the Output . . . . .	46
3.3	Reading from File . . . . .	47
3.4	Reading from the Command Line . . . . .	49
3.5	Tips: Controlling Output Format . . . . .	50
3.6	Exercises . . . . .	51
<b>4</b>	<b>Pointers . . . . .</b>	<b>55</b>
4.1	Pointers and the Computer's Memory . . . . .	55
4.1.1	Addresses . . . . .	55
4.1.2	Pointer Variables . . . . .	56
4.1.3	Example Use of Pointers . . . . .	56
4.1.4	Warnings on the Use of Pointers . . . . .	57
4.2	Dynamic Allocation of Memory for Arrays . . . . .	58
4.2.1	Vectors . . . . .	59
4.2.2	Matrices . . . . .	60
4.2.3	Irregularly Sized Matrices . . . . .	61
4.3	Tips: Pointers . . . . .	62
4.3.1	Tip 1: Pointer Aliasing . . . . .	62
4.3.2	Tip 2: Safe Dynamic Allocation . . . . .	63
4.3.3	Tip 3: Every <code>new</code> Has a <code>delete</code> . . . . .	63
4.4	Exercises . . . . .	64
<b>5</b>	<b>Blocks, Functions and Reference Variables . . . . .</b>	<b>65</b>
5.1	Blocks . . . . .	65
5.2	Functions . . . . .	66
5.2.1	Simple Functions . . . . .	66
5.2.2	Returning Pointer Variables from a Function . . . . .	69
5.2.3	Use of Pointers as Function Arguments . . . . .	70
5.2.4	Sending Arrays to Functions . . . . .	71



5.2.5	Example: A Function to Calculate the Scalar Product of Two Vectors . . . . .	73
5.3	Reference Variables . . . . .	74
5.4	Default Values for Function Arguments . . . . .	75
5.5	Function Overloading . . . . .	76
5.6	Declaring Functions Without Prototypes . . . . .	78
5.7	Function Pointers . . . . .	79
5.8	Recursive Functions . . . . .	81
5.9	Modules . . . . .	82
5.10	Tips: Code Documentation . . . . .	83
5.11	Exercises . . . . .	85
<b>6</b>	<b>An Introduction to Classes . . . . .</b>	<b>87</b>
6.1	The <i>Raison d'Être</i> for Classes . . . . .	87
6.1.1	Problems That May Arise When Using Modules . . . . .	88
6.1.2	Abstraction, Encapsulation and Modularity Properties of Classes . . . . .	88
6.2	A First Example Simple Class: A Class of Books . . . . .	89
6.2.1	Basic Features of Classes . . . . .	89
6.2.2	Header Files . . . . .	91
6.2.3	Setting and Accessing Variables . . . . .	92
6.2.4	Compiling Multiple Files . . . . .	94
6.2.5	Access Privileges . . . . .	96
6.2.6	Including Function Implementations in Header Files . . . . .	97
6.2.7	Constructors and Destructors . . . . .	98
6.2.8	Pointers to Classes . . . . .	103
6.3	The <code>friend</code> Keyword . . . . .	103
6.4	A Second Example Class: A Class of Complex Numbers . . . . .	105
6.4.1	Operator Overloading . . . . .	105
6.4.2	The Class of Complex Numbers . . . . .	106
6.5	Some Additional Remarks on Operator Overloading . . . . .	112
6.6	Tips: Coding to a Standard . . . . .	112
6.7	Exercises . . . . .	114
<b>7</b>	<b>Inheritance and Derived Classes . . . . .</b>	<b>117</b>
7.1	Inheritance, Extensibility and Polymorphism . . . . .	117
7.2	Example: A Class of E-books Derived from a Class of Books . . . . .	118
7.3	Access Privileges for Derived Classes . . . . .	120
7.4	Classes Derived from Derived Classes . . . . .	121
7.5	Run-Time Polymorphism . . . . .	122
7.6	The Abstract Class Pattern . . . . .	124
7.7	Tips: Using a Debugger . . . . .	126
7.8	Exercises . . . . .	127
<b>8</b>	<b>Templates . . . . .</b>	<b>131</b>
8.1	Templates to Control Dimensions and Verify Sizes . . . . .	131

8.2	Templates for Polymorphism . . . . .	133
8.3	A Brief Survey of the Standard Template Library . . . . .	134
8.3.1	Vectors . . . . .	134
8.3.2	Sets . . . . .	137
8.4	Tips: Template Compilation . . . . .	139
8.5	Exercises . . . . .	140
<b>9</b>	<b>Errors and Exceptions . . . . .</b>	<b>141</b>
9.1	Preconditions . . . . .	142
9.1.1	Example: Two Implementations of a Graphics Function . . . . .	142
9.2	Three Levels of Errors . . . . .	143
9.3	Introducing the Exception . . . . .	144
9.4	Using Exceptions . . . . .	145
9.5	Tips: Test-Driven Development . . . . .	146
9.6	Exercises . . . . .	147
<b>10</b>	<b>Developing Classes for Linear Algebra Calculations . . . . .</b>	<b>151</b>
10.1	Requirements of the Linear Algebra Classes . . . . .	151
10.2	Constructors and Destructors . . . . .	156
10.2.1	The Default Constructor . . . . .	156
10.2.2	The Copy Constructor . . . . .	156
10.2.3	A Specialised Constructor . . . . .	157
10.2.4	Destructor . . . . .	157
10.3	Accessing Private Class Members . . . . .	157
10.3.1	Accessing the Size of a Vector . . . . .	158
10.3.2	Overloading the Square Bracket Operator . . . . .	158
10.3.3	Read-Only Access to Vector Entries . . . . .	158
10.3.4	Overloading the Round Bracket Operator . . . . .	158
10.4	Operator Overloading for Vector Operations . . . . .	158
10.4.1	The Assignment Operator . . . . .	159
10.4.2	Unary Operators . . . . .	159
10.4.3	Binary Operators . . . . .	159
10.5	Functions . . . . .	159
10.5.1	Members Versus Friends . . . . .	159
10.6	Tips: Memory Debugging Tools . . . . .	160
10.7	Exercises . . . . .	161
<b>11</b>	<b>An Introduction to Parallel Programming Using MPI . . . . .</b>	<b>165</b>
11.1	Distributed Memory Architectures . . . . .	165
11.2	Installing MPI . . . . .	167
11.3	A First Program Using MPI . . . . .	167
11.3.1	Essential MPI Functions . . . . .	168
11.3.2	Compiling and Running MPI Code . . . . .	169
11.4	Basic MPI Communication . . . . .	171
11.4.1	Point-to-Point Communication . . . . .	171
11.4.2	Collective Communication . . . . .	174

---

11.5	Example MPI Applications . . . . .	180
11.5.1	Summation of Series . . . . .	180
11.5.2	Parallel Linear Algebra . . . . .	182
11.6	Tips: Debugging a Parallel Program . . . . .	186
11.6.1	Tip 1: Make an Abstract Program . . . . .	186
11.6.2	Tip 2: Datatype Mismatch . . . . .	186
11.6.3	Tip 3: Intermittent Deadlock . . . . .	187
11.6.4	Tip 4: Almost Collective Communication . . . . .	187
11.7	Exercises . . . . .	188
<b>12</b>	<b>Designing Object-Oriented Numerical Libraries . . . . .</b>	<b>193</b>
12.1	Developing the Library for Ordinary Differential Equations . . . . .	194
12.1.1	Model Problems . . . . .	194
12.1.2	Finite Difference Approximation to Derivatives . . . . .	195
12.1.3	Application of Finite Difference Methods to Boundary Value Problems . . . . .	197
12.1.4	Concluding Remarks on Boundary Value Problems in One Dimension . . . . .	199
12.2	Designing a Library for Solving Boundary Value Problems . . . . .	200
12.2.1	The Class <code>SecondOrderOde</code> . . . . .	200
12.2.2	The Class <code>BoundaryConditions</code> . . . . .	201
12.2.3	The Class <code>FiniteDifferenceGrid</code> . . . . .	202
12.2.4	The Class <code>BvpOde</code> . . . . .	203
12.2.5	Using the Class <code>BvpOde</code> . . . . .	205
12.3	Extending the Library to Two Dimensions . . . . .	205
12.3.1	Model Problem for Two Dimensions . . . . .	207
12.3.2	Finite Difference Methods for Boundary Value Problems in Two Dimensions . . . . .	207
12.3.3	Setting Up the Linear System for the Model Problem . . . . .	209
12.3.4	Developing the Classes Required . . . . .	210
12.4	Tips: Using Well-Written Libraries . . . . .	210
12.5	Exercises . . . . .	211
<b>Appendix A</b>	<b>Linear Algebra . . . . .</b>	<b>213</b>
A.1	Vectors and Matrices . . . . .	213
A.1.1	Operations Between Vectors and Matrices . . . . .	214
A.1.2	The Scalar Product of Two Vectors . . . . .	215
A.1.3	The Determinant and the Inverse of a Matrix . . . . .	215
A.1.4	Eigenvalues and Eigenvectors of a Matrix . . . . .	216
A.1.5	Vector and Matrix Norms . . . . .	216
A.2	Systems of Linear Equations . . . . .	217
A.2.1	Gaussian Elimination . . . . .	217
A.2.2	The Thomas Algorithm . . . . .	222
A.2.3	The Conjugate Gradient Method . . . . .	222

---

<b>Appendix B Other Programming Constructs You Might Meet</b> . . . . .	225
B.1 C Style Output . . . . .	225
B.2 C Style Dynamic Memory Allocation . . . . .	226
B.3 Ternary ? : Operator . . . . .	226
B.4 Using Namespace . . . . .	227
B.5 Structures . . . . .	228
B.6 Multiple Inheritance . . . . .	228
B.7 Class Initialisers . . . . .	229
<b>Appendix C Solutions to Exercises</b> . . . . .	231
C.1 Matrix and Linear System Classes . . . . .	231
C.2 ODE Solver Library . . . . .	240
<b>Further Reading</b> . . . . .	245
Mathematical Methods and Linear Algebra . . . . .	245
C++ Programming . . . . .	245
The Message-Passing Interface (MPI) . . . . .	245
<b>Index</b> . . . . .	247