

- [Table of Contents](#)

C++ Templates: The Complete Guide

By [David Vandevoorde](#), [Nicolai M. Josuttis](#)

Publisher	: Addison Wesley
Pub Date	: November 12, 2002
ISBN	: 0-201-73484-2
Pages	: 552

[Copyright](#)

[Preface](#)

[Acknowledgments](#)

[Nico's Acknowledgments](#)

[David's Acknowledgments](#)

[Chapter 1. About This Book](#)

[Section 1.1. What You Should Know Before Reading This Book](#)

[Section 1.2. Overall Structure of the Book](#)

[Section 1.3. How to Read This Book](#)

[Section 1.4. Some Remarks About Programming Style](#)

[Section 1.5. The Standard versus Reality](#)

[Section 1.6. Example Code and Additional Informations](#)

[Section 1.7. Feedback](#)

[Part I: The Basics](#)

[Chapter 2. Function Templates](#)

[Section 2.1. A First Look at Function Templates](#)

[Section 2.2. Argument Deduction](#)

[Section 2.3. Template Parameters](#)

[Section 2.4. Overloading Function Templates](#)

[Section 2.5. Summary](#)

[Chapter 3. Class Templates](#)

[Section 3.1. Implementation of Class Template Stack](#)

[Section 3.2. Use of Class Template Stack](#)

[Section 3.3. Specializations of Class Templates](#)

[Section 3.4. Partial Specialization](#)

[Section 3.5. Default Template Arguments](#)

[Section 3.6. Summary](#)

[Chapter 4. Nontype Template Parameters](#)

[Section 4.1. Nontype Class Template Parameters](#)
[Section 4.2. Nontype Function Template Parameters](#)
[Section 4.3. Restrictions for Nontype Template Parameters](#)
[Section 4.4. Summary](#)

[Chapter 5. Tricky Basics](#)

[Section 5.1. Keyword typename](#)
[Section 5.2. Using this->](#)
[Section 5.3. Member Templates](#)
[Section 5.4. Template Template Parameters](#)
[Section 5.5. Zero Initialization](#)
[Section 5.6. Using String Literals as Arguments for Function Templates](#)
[Section 5.7. Summary](#)

[Chapter 6. Using Templates in Practice](#)

[Section 6.1. The Inclusion Model](#)
[Section 6.2. Explicit Instantiation](#)
[Section 6.3. The Separation Model](#)
[Section 6.4. Templates and inline](#)
[Section 6.5. Precompiled Headers](#)
[Section 6.6. Debugging Templates](#)
[Section 6.7. Afternotes](#)
[Section 6.8. Summary](#)

[Chapter 7. Basic Template Terminology](#)

[Section 7.1. "Class Template" or "Template Class"?](#)
[Section 7.2. Instantiation and Specialization](#)
[Section 7.3. Declarations versus Definitions](#)
[Section 7.4. The One-Definition Rule](#)
[Section 7.5. Template Arguments versus Template Parameters](#)

[Part II: Templates in Depth](#)

[Chapter 8. Fundamentals in Depth](#)

[Section 8.1. Parameterized Declarations](#)
[Section 8.2. Template Parameters](#)
[Section 8.3. Template Arguments](#)
[Section 8.4. Friends](#)
[Section 8.5. Afternotes](#)

[Chapter 9. Names in Templates](#)

[Section 9.1. Name Taxonomy](#)
[Section 9.2. Looking Up Names](#)
[Section 9.3. Parsing Templates](#)
[Section 9.4. Derivation and Class Templates](#)
[Section 9.5. Afternotes](#)

[Chapter 10. Instantiation](#)

[Section 10.1. On-Demand Instantiation](#)

[Section 10.2. Lazy Instantiation](#)

[Section 10.3. The C++ Instantiation Model](#)

[Section 10.4. Implementation Schemes](#)

[Section 10.5. Explicit Instantiation](#)

[Section 10.6. Afternotes](#)

[Chapter 11. Template Argument Deduction](#)

[Section 11.1. The Deduction Process](#)

[Section 11.2. Deduced Contexts](#)

[Section 11.3. Special Deduction Situations](#)

[Section 11.4. Allowable Argument Conversions](#)

[Section 11.5. Class Template Parameters](#)

[Section 11.6. Default Call Arguments](#)

[Section 11.7. The Barton-Nackman Trick](#)

[Section 11.8. Afternotes](#)

[Chapter 12. Specialization and Overloading](#)

[Section 12.1. When "Generic Code" Doesn't Quite Cut It](#)

[Section 12.2. Overloading Function Templates](#)

[Section 12.3. Explicit Specialization](#)

[Section 12.4. Partial Class Template Specialization](#)

[Section 12.5. Afternotes](#)

[Chapter 13. Future Directions](#)

[Section 13.1. The Angle Bracket Hack](#)

[Section 13.2. Relaxed typename Rules](#)

[Section 13.3. Default Function Template Arguments](#)

[Section 13.4. String Literal and Floating-Point Template Arguments](#)

[Section 13.5. Relaxed Matching of Template Template Parameters](#)

[Section 13.6. Typedef Templates](#)

[Section 13.7. Partial Specialization of Function Templates](#)

[Section 13.8. The typeof Operator](#)

[Section 13.9. Named Template Arguments](#)

[Section 13.10. Static Properties](#)

[Section 13.11. Custom Instantiation Diagnostics](#)

[Section 13.12. Overloaded Class Templates](#)

[Section 13.13. List Parameters](#)

[Section 13.14. Layout Control](#)

[Section 13.15. Initializer Deduction](#)

[Section 13.16. Function Expressions](#)

[Section 13.17. Afternotes](#)

[Part III: Templates and Design](#)

[Chapter 14. The Polymorphic Power of Templates](#)

[Section 14.1. Dynamic Polymorphism](#)
[Section 14.2. Static Polymorphism](#)
[Section 14.3. Dynamic versus Static Polymorphism](#)
[14.4 New Forms of Design Patterns](#)
[Section 14.5. Generic Programming](#)
[Section 14.6. Afternotes](#)

[Chapter 15. Traits and Policy Classes](#)
[Section 15.1. An Example: Accumulating a Sequence](#)
[Section 15.2. Type Functions](#)
[Section 15.3. Policy Traits](#)
[Section 15.4. Afternotes](#)

[Chapter 16. Templates and Inheritance](#)
[Section 16.1. Named Template Arguments](#)
[Section 16.2. The Empty Base Class Optimization \(EBCO\)](#)
[Section 16.3. The Curiously Recurring Template Pattern \(CRTP\)](#)
[Section 16.4. Parameterized Virtuality](#)
[Section 16.5. Afternotes](#)

[Chapter 17. Metaprograms](#)
[Section 17.1. A First Example of a Metaprogram](#)
[Section 17.2. Enumeration Values versus Static Constants](#)
[Section 17.3. A Second Example: Computing the Square Root](#)
[Section 17.4. Using Induction Variables](#)
[Section 17.5. Computational Completeness](#)
[Section 17.6. Recursive Instantiation versus Recursive Template Arguments](#)
[Section 17.7. Using Metaprograms to Unroll Loops](#)
[Section 17.8. Afternotes](#)

[Chapter 18. Expression Templates](#)
[Section 18.1. Temporaries and Split Loops](#)
[Section 18.2. Encoding Expressions in Template Arguments](#)
[Section 18.3. Performance and Limitations of Expression Templates](#)
[Section 18.4. Afternotes](#)

[Part IV: Advanced Applications](#)

[Chapter 19. Type Classification](#)
[Section 19.1. Determining Fundamental Types](#)
[Section 19.2. Determining Compound Types](#)
[Section 19.3. Identifying Function Types](#)
[Section 19.4. Enumeration Classification with Overload Resolution](#)
[Section 19.5. Determining Class Types](#)
[Section 19.6. Putting It All Together](#)
[Section 19.7. Afternotes](#)

[Chapter 20. Smart Pointers](#)

[Section 20.1. Holders and Trules](#)

[Section 20.2. Reference Counting](#)

[Section 20.3. Afternotes](#)

[Chapter 21. Tuples](#)

[Section 21.1. Duos](#)

[Section 21.2. Recursive Duos](#)

[Section 21.3. Tuple Construction](#)

[Section 21.4. Afternotes](#)

[Chapter 22. Function Objects and Callbacks](#)

[Section 22.1. Direct, Indirect, and Inline Calls](#)

[Section 22.2. Pointers and References to Functions](#)

[Section 22.3. Pointer-to-Member Functions](#)

[Section 22.4. Class Type Functors](#)

[Section 22.5. Specifying Functors](#)

[Section 22.6. Introspection](#)

[Section 22.7. Function Object Composition](#)

[Section 22.8. Value Binders](#)

[Functor Operations: A Complete Implementation](#)

[Section 22.10. Afternotes](#)

[Appendix A. The One-Definition Rule](#)

[Section A.1. Translation Units](#)

[Section A.2. Declarations and Definitions](#)

[Section A.3. The One-Definition Rule in Detail](#)

[Appendix B. Overload Resolution](#)

[Section B.1. When Does Overload Resolution Kick In?](#)

[Section B.2. Simplified Overload Resolution](#)

[Section B.3. Overloading Details](#)

[Bibliography](#)

[Newsgroups](#)

[Books and Web Sites](#)

[Glossary](#)