
Effective Modern C++

Scott Meyers

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'REILLY®

Table of Contents

From the Publisher.....	xi
Acknowledgments.....	xiii
Introduction.....	1
1. Deducing Types.....	9
Item 1: Understand template type deduction.	9
Item 2: Understand <code>auto</code> type deduction.	18
Item 3: Understand <code>decltype</code> .	23
Item 4: Know how to view deduced types.	30
2. <code>auto</code>.....	37
Item 5: Prefer <code>auto</code> to explicit type declarations.	37
Item 6: Use the explicitly typed initializer idiom when <code>auto</code> deduces undesired types.	43
3. Moving to Modern C++.....	49
Item 7: Distinguish between <code>()</code> and <code>{}</code> when creating objects.	49
Item 8: Prefer <code>nullptr</code> to <code>0</code> and <code>NULL</code> .	58
Item 9: Prefer alias declarations to <code>typedefs</code> .	63
Item 10: Prefer <code>scoped enums</code> to <code>unscoped enums</code> .	67
Item 11: Prefer <code>deleted functions</code> to <code>private undefined ones</code> .	74
Item 12: Declare overriding functions <code>override</code> .	79
Item 13: Prefer <code>const iterators</code> to <code>iterators</code> .	86
Item 14: Declare functions <code>noexcept</code> if they won't emit exceptions.	90
Item 15: Use <code>constexpr</code> whenever possible.	97

Item 16: Make const member functions thread safe.	103
Item 17: Understand special member function generation.	109
4. Smart Pointers.	117
Item 18: Use <code>std::unique_ptr</code> for exclusive-ownership resource management.	118
Item 19: Use <code>std::shared_ptr</code> for shared-ownership resource management.	125
Item 20: Use <code>std::weak_ptr</code> for <code>std::shared_ptr</code> -like pointers that can dangle.	134
Item 21: Prefer <code>std::make_unique</code> and <code>std::make_shared</code> to direct use of <code>new</code> .	139
Item 22: When using the Pimpl Idiom, define special member functions in the implementation file.	147
5. Rvalue References, Move Semantics, and Perfect Forwarding.	157
Item 23: Understand <code>std::move</code> and <code>std::forward</code> .	158
Item 24: Distinguish universal references from rvalue references.	164
Item 25: Use <code>std::move</code> on rvalue references, <code>std::forward</code> on universal references.	168
Item 26: Avoid overloading on universal references.	177
Item 27: Familiarize yourself with alternatives to overloading on universal references.	184
Item 28: Understand reference collapsing.	197
Item 29: Assume that move operations are not present, not cheap, and not used.	203
Item 30: Familiarize yourself with perfect forwarding failure cases.	207
6. Lambda Expressions.	215
Item 31: Avoid default capture modes.	216
Item 32: Use <code>init</code> capture to move objects into closures.	224
Item 33: Use <code>decltype</code> on <code>auto&&</code> parameters to <code>std::forward</code> them.	229
Item 34: Prefer lambdas to <code>std::bind</code> .	232
7. The Concurrency API.	241
Item 35: Prefer task-based programming to thread-based.	241
Item 36: Specify <code>std::launch::async</code> if asynchronicity is essential.	245
Item 37: Make <code>std::threads</code> unjoinable on all paths.	250
Item 38: Be aware of varying thread handle destructor behavior.	258
Item 39: Consider <code>void</code> futures for one-shot event communication.	262

Item 40: Use <code>std::atomic</code> for concurrency, <code>volatile</code> for special memory.	271
8. Tweaks	281
Item 41: Consider pass by value for copyable parameters that are cheap to move and always copied.	281
Item 42: Consider emplacement instead of insertion.	292
Index	303

Using Code

[Introduction](#)
[Chapter 1](#)
[Chapter 2](#)
[Chapter 3](#)
[Chapter 4](#)
[Chapter 5](#)
[Chapter 6](#)
[Chapter 7](#)
[Chapter 8](#)
[Chapter 9](#)
[Chapter 10](#)
[Chapter 11](#)
[Chapter 12](#)
[Chapter 13](#)
[Chapter 14](#)
[Chapter 15](#)
[Chapter 16](#)
[Chapter 17](#)
[Chapter 18](#)
[Chapter 19](#)
[Chapter 20](#)
[Chapter 21](#)
[Chapter 22](#)
[Chapter 23](#)
[Chapter 24](#)
[Chapter 25](#)
[Chapter 26](#)
[Chapter 27](#)
[Chapter 28](#)
[Chapter 29](#)
[Chapter 30](#)
[Chapter 31](#)
[Chapter 32](#)
[Chapter 33](#)
[Chapter 34](#)
[Chapter 35](#)
[Chapter 36](#)
[Chapter 37](#)
[Chapter 38](#)
[Chapter 39](#)
[Chapter 40](#)
[Chapter 41](#)
[Chapter 42](#)
[Chapter 43](#)
[Chapter 44](#)
[Chapter 45](#)
[Chapter 46](#)
[Chapter 47](#)
[Chapter 48](#)
[Chapter 49](#)
[Chapter 50](#)
[Chapter 51](#)
[Chapter 52](#)
[Chapter 53](#)
[Chapter 54](#)
[Chapter 55](#)
[Chapter 56](#)
[Chapter 57](#)
[Chapter 58](#)
[Chapter 59](#)
[Chapter 60](#)
[Chapter 61](#)
[Chapter 62](#)
[Chapter 63](#)
[Chapter 64](#)
[Chapter 65](#)
[Chapter 66](#)
[Chapter 67](#)
[Chapter 68](#)
[Chapter 69](#)
[Chapter 70](#)
[Chapter 71](#)
[Chapter 72](#)
[Chapter 73](#)
[Chapter 74](#)
[Chapter 75](#)
[Chapter 76](#)
[Chapter 77](#)
[Chapter 78](#)
[Chapter 79](#)
[Chapter 80](#)
[Chapter 81](#)
[Chapter 82](#)
[Chapter 83](#)
[Chapter 84](#)
[Chapter 85](#)
[Chapter 86](#)
[Chapter 87](#)
[Chapter 88](#)
[Chapter 89](#)
[Chapter 90](#)
[Chapter 91](#)
[Chapter 92](#)
[Chapter 93](#)
[Chapter 94](#)
[Chapter 95](#)
[Chapter 96](#)
[Chapter 97](#)
[Chapter 98](#)
[Chapter 99](#)
[Chapter 100](#)