

---

# Effective Modern C++

*Scott Meyers*

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

**O'REILLY®**

## Effective Modern C++

by Scott Meyers

Copyright © 2015 Scott Meyers. All rights reserved.

Printed in the Canada.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editor:** Rachel Roumeliotis

**Production Editor:** Melanie Yarbrough

**Copyeditor:** Jasmine Kwityn

**Proofreader:** Charles Roumeliotis

**Indexer:** Scott Meyers

**Interior Designer:** David Futato

**Cover Designer:** Ellie Volkhausen

**Illustrator:** Rebecca Demarest

November 2014: First Edition

### Revision History for the First Edition

2014-11-07: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781491903995> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Effective Modern C++*, the cover image of a Rose-crowned Fruit Dove, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-90399-5

[TI]

---

# Table of Contents

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

Item 16: Make <code>const</code> member functions thread safe.	103
Item 17: Understand special member function generation.	109
<b>4. Smart Pointers.....</b>	<b>117</b>
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
<b>5. Rvalue References, Move Semantics, and Perfect Forwarding.....</b>	<b>157</b>
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
<b>6. Lambda Expressions.....</b>	<b>215</b>
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&amp;&amp;</code> parameters to <code>std::forward</code> them.	229
Item 34: Prefer lambdas to <code>std::bind</code> .	232
<b>7. The Concurrency API.....</b>	<b>241</b>
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 void futures for one-shot event communication.	262

Item 40: Use <code>std::atomic</code> for concurrency, <code>volatile</code> for special memory.	271
<b>8. Tweaks.....</b>	<b>281</b>
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
<b>Index.....</b>	<b>303</b>