
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 auto type deduction.	18
Item 3: Understand decltype.	23
Item 4: Know how to view deduced types.	30
2. auto.....	37
Item 5: Prefer auto to explicit type declarations.	37
Item 6: Use the explicitly typed initializer idiom when auto deduces undesired types.	43
3. Moving to Modern C++.....	49
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
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 capture</code> 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 void 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