

Index

- immutable types, optimizing, 1167–1170
- imperative programming, 959
- implementation defined, 1093
 - alignments, 168–169
 - NULL macro, 100
 - opaque enumerations, 660
- implementation inheritance, avoiding boilerplate code with, 540–541
- implementation-defined behavior
 - enum** class, 335
 - limits on, 295
 - unrecognized attributes, 12, 18–19
- implementation-defined types, 202, 501
- implicit **const**-qualification, 300
- implicit constructors, inheriting, 546–549
- implicit conversion, 66, 223n3
 - to arithmetic types, avoiding, 337–339
 - preventing, 55–56, 201
- implicit generation of special member functions, 44–45
- implicit moves
 - disabling, 244–246
 - in **return** statements, 735–737
- implicitly captured, 582–583
- implicitly declared, 522
- implicitly declared default constructors, 568–570
- implicitly movable entities, 735n13
- in contract, 1122–1123
- in place, 734
- indentation of string literals, 112–113
- indeterminate values, 435, 493–497
- infallible, 1118
- infallible implementation, 1118–1123
- inheritance
 - improving concrete class performance, 1015–1017
 - preventing with **final** contextual keyword, 1008, 1012
- inheriting constructors. *See also* default member initializers; defaulted functions; delegating constructors; deleted functions; forwarding references; **override** member-function specifier; variadic templates
 - annoyances, 549–552
 - access levels same as in base class, 549–551
 - cannot select individually, 549
 - flawed initial specification, 551–552
 - description of, 535–539
 - potential pitfalls, 546–549
 - implicit constructors, 546–549
 - new constructors in base class alters behavior, 546
 - use cases, 539–545
 - implementation inheritance, avoiding boilerplate code, 540–541
 - reusable functionality through mix-ins, 545
 - strong **typedef** implementation, 541–544
 - structural inheritance, avoiding boilerplate code, 540
- init capture, 986
- initialization. *See also* aggregate initialization; braced initialization; copy initialization; copy list initialization; default initialization; default member initializers; direct initialization; direct list initialization; list initialization; **std::initializer_list**; uniform initialization; value initialization
 - of bit fields, 329n4
 - concurrent, 68–69
 - constant, 75
 - enforcing with PassKey idiom, 1036–1038
 - recursive, 77–78, 163–165
 - of simple structs, 322
 - subject, inconsistency in, 326–328
 - of subobjects, inconsistency in, 326–328
 - thread-safe function-scope static variables, 68–69
 - trivial default, 1087
 - of variables, 200
 - wrapping in factory functions, 389–390
- initializer lists. *See* **std::initializer_list**
- initializer_list**. *See* **std::initializer_list**
- initializers, undefined behavior with **constexpr** variables, 306–307
- inline namespace** sets, 1056
- inline namespaces**. *See also* **alignas** specifier
 - annoyances, 1079–1082
 - code factoring, impeding, 1079–1082
 - one-to-one relationship with namespaces, 1082
 - argument-dependent lookup (ADL) interoperability, 1058–1059
 - class template specialization, 1059–1061
 - description of, 1055–1062
 - duplicate names, loss of access to, 1056–1058
 - further reading for, 1083
 - potential pitfalls, 1076–1079
 - inconsistent use of **inline** keyword, 1079
 - lack of scalability, 1076–1077
 - library evolution, 1077–1079
 - reopening, 1061–1062