

Index

- translation unit (TU)
 - opaque enumerations, 660
 - thread-safe function-scope static variables, 71
- translation-lookaside buffer (TLB), 182n13
- transparently nested namespaces. *See* inline namespaces
- trivial, 38
- trivial classes, 521
- trivial constructors, 273–274, 408n4
- trivial copy constructors, 470, 528n62
- trivial copy operation, 483, 733, 812
- trivial copy-assignment operator, 470
- trivial default constructors, 461
- trivial default initialization, 1087
- trivial destructibility, 468–469
- trivial destructors, 408n4
- trivial move constructors, 484, 528n62
- trivial move operation, 733
- trivial operations, 33
- trivial types
 - in C++17, 425n7
 - fixed-capacity string elements, 476–479
 - future direction of PODs, 438–439
 - generalized PODs, 401, 416–417, 425–429
 - preserving, 39–40
 - requiring, 480–482
 - special member functions and, 1012
 - subcategories, 429–436
 - union membership and, 1174
- triviality, loss of, 329–330
- trivially constructible, 80n7
 - POD types, 431–432
 - secure buffers, 460–462
- trivially copy assignable, 486–487
- trivially copy constructible, 488
- trivially copyable, 39, 41–42
 - C++ Standard not stabilized, 521–527
 - fixed-capacity strings, 470–475
 - ineligible use of `std::memcpy`, 497–501
 - `memcpy` usage on `const` or reference subobjects, 489–493
 - naive copying other than `std::memcpy`, 501–505
 - POD types, 401, 434–436
 - sloppy terminology, 488–489
 - wrong usage of type traits, 482–488
- trivially copyable class, 521
- trivially copyable types, 468
- trivially default constructible, 401, 430–436
- trivially destructible
 - compile-time constructible, literal types, 462–464
 - `constexpr` variables, 305
 - POD types, 402, 430–434
 - reducing code size, 1104
 - sloppy terminology, 488–489
- trivially destructible types in C++20, 430n9
- true sharing, 183n15
- tuples, 932–937, 975–976
- type aliases. *See also* inheriting constructors; trailing return
 - befriending as customization point, 1034–1036
 - creating with `using` declarations, 133–137
 - description of, 133–134
 - exception specifications and, 1090, 1147
 - use cases, 134–137
 - binding arguments to template parameters, 135–136
 - simplified `typedef` declarations, 134–135
 - type trait notation, 136–137
- ~~type categories, 837, 843~~
- type deduction
 - forwarding references, 379–380
 - of `std::initializer_list`, 559–561
- type erasure, 602
- type identifiers as `alignas` specifier argument, 172
- type inference, 193
- type lists, 963
- type parameter packs, 903
- type punning, 401
- type safety, 100–101
- type suffix, 837
- type template parameter packs, 880–884
- type template parameters, 902
- type traits, 436–438
 - in C++17, 651n12
 - notation, 136–137
 - reducing verbosity, 161–163
 - `static_assert`, 119
 - `std::is_lvalue_reference`, 378
 - as unreliable, 527–528
 - wrong usage, 482–488
- `<type_traits>` header, 1014
- type-consistency, explicit expression of, 27–28, 28n1
- `typedef`. *See also* aliases
 - capturing results of `decltype` expressions in, 31
 - in `<cstdint>`, 92
 - strong implementation, 541–544
- typename disambiguator, 382n1
- typename specifiers, 1032
- typenames
 - explicit, 26–27
 - in `friend` declarations, 1033n1

Index

- types. *See also* POD types; trivial types; type aliases; type safety; type traits; user-defined types (UDTs); value-semantic types (VSTs)
 - as **alignof** argument, 193–194
 - function pointers and, 265–266
 - historical perspective on, 93–94
 - literal, 278–284
 - local/unnamed
 - description of, 83–84
 - use cases, 84–87
 - long long**
 - description of, 89
 - further reading for, 92
 - potential pitfalls, 91–92
 - use cases, 89–91
 - redundant repetition, avoiding, 200–201
 - relative sizes of, 91–92
 - scalar
 - aggregate initialization, 222
 - copy initialization, 235–236
 - initialization, 217
 - trailing return. *See also* **decltype**; deduced return type
 - description of, 124–126
 - further reading for, 128
 - inferring type of, 28
 - use cases, 126–128
 - underlying types (UTs)
 - description of, 829–830
 - further reading for, 834
 - potential pitfalls, 832–833
 - use cases, 830–832
 - union
 - description of, 1174–1177
 - further reading for, 1181
 - potential pitfalls, 1180
 - use cases, 1177–1180
 - variant, 937–948
- U**
- UDL operator templates, 841, 849–851, 870
- UDL operators, 840–842
 - cooked, 843–845
 - raw, 845–849
 - templates, 849–851
- UDL suffix, 837
- UDL type categories, 843
- UDTs. *See* user-defined types (UDTs)
- unconditional exception specifications, 1085–1089
- undefined behavior (UB), 1024n10, 1077, 1104, 1175
 - attributes and, 18–19
 - auto** return-type deduction, 1187
 - constexpr** variable initializers, 306–307
 - contract guarantees, 1115
 - delegating constructors, 50n2
 - diagnosing at compile time, 312–314
 - friend** declarations, 1049
 - generalized PODs, 401
 - long long** integral type, 90
 - [[noreturn]] attribute, 97
 - range-based **for** loops, 692
 - rvalue references, 715
 - thread-safe function-scope static variables, 70
 - uninitialized values, 218
 - union type and, 1180
- undefined symbol links, 1068n4
- undefined symbols, 363
- underlying types (UTs)
 - constexpr** variables, 308–309
 - description of, 829–830
 - enum** class, 337
 - enumerations, 333–334
 - further reading for, 834
 - opaque enumerations, 660
 - potential pitfalls, 832–833
 - Unicode string literals, 131
 - use cases, 830–832
- underspecifying alignment, 176
- unevaluated contexts, `std::declval` used in, 31, 1132
- unevaluated operands, 615
- Unicode string literals
 - description of, 129–130
 - potential pitfalls, 130–132
 - embedding Unicode graphemes, 130–131
 - library support, lack of, 131
 - UTF-8, problematic treatment of, 131–132
 - use cases, 130
- unification, 901
- uniform initialization, 215
 - in factory functions, 239–241
 - in generic code, 238–239
 - member initialization in generic code, 241–242
- union type
 - description of, 1174–1177
 - discriminated unions, 937–948
 - further reading for, 1181
 - misuse of, 505–506
 - potential pitfalls, 1180
 - use cases, 1177–1180
 - vertical encoding within, 439–448
- unions
 - default member initializers and, 320–321
 - final** contextual keyword in, 1013
- unique object address, 418