

# Index

- , (comma operator), 39–40
  - ?: (conditional operator), 15–16, 40–41
  - [ ] (allocating and deleting arrays), 35, 36, 168
  - () (allocating arrays), 35
  - > (arrow operator), 58–60, 257–258
  - [] (index operator), 16–17
  - && (logical operator), 40
  - || (logical operator), 40
  - <<< (Sergeant operator), 48–49
- 
- A**
- access protection
    - Bridge pattern, 21–22
    - vs. data abstraction, 241–242
    - description, 19
    - inheritance, 23, 277–280
    - naming conventions, 23
    - vs. visibility, 19–23
  - accessor functions. *See* get/set interfaces.
  - ACM Code of Ethics...*, 32
  - acquaintance vs. aggregation, 253–258
  - acronyms, 25–26
  - Acyclic Visitor pattern, 306
  - addresses
    - arithmetic errors, 77–78, 89, 101, 286
    - base class subobjects, 206–208
    - members. *See* pointers, to members.
    - of a non-lvalue return, 267
  - adolescent behavior, 31–33
  - aggregation vs. acquaintance, 253–258
  - Alexandrescu, Andrei, xv
  - algorithms, variant and invariant, 212–214
  - aliases
    - aggregation/acquaintance relationships, 253
    - references as, 10–13, 112–115
  - allocation failure, 171–173
  - ampersands (&&) logical operator, 40
  - anonymous namespace, 55
  - anonymous temporaries
    - function object for pass by value, 109, 111
    - initialization of a reference formal argument, 107
    - initialize reference to const, 112
    - lifetime, 110–111
    - result of a postfix ++ or --, 266
    - as a result of copy initialization, 153–154
    - throwing, 181–182
  - array names vs. constant pointers, 87
  - arrays
    - of arrays, 52, 87–88
    - of class objects, 271–273
    - confused with initializers, 35–36
    - freeing, 167–170
    - migrating type-qualifiers, 52
    - pointer-to-multidimensional array conversions, 87–88
    - references to, 12
    - vs. vectors, 168

arrow operator (->), 58–60, 257–258  
 assert macro, 72–74  
 assertions, side effects, 72–74  
 assignment vs. initialization, 125–129,  
 139–141  
 associativity  
   and precedence, 42  
   problems, 44–45  
 auto\_ptr template, 28–29, 195–197

---

## B

base class subobject  
   addresses, 206–208  
   initialization, 142–147, 147–150,  
   218–219  
 base class types, arrays of class objects,  
 271–273  
 base language. *See* C++ base language.  
 battleship, in pencil cup, 295  
 binding  
   dynamic, 200  
   reference to function, 13  
   reference to lvalue, 11, 112–115  
 Bridge pattern, 21–22

---

## C

C++ base language  
   conditional operator, 15–16  
   fallthrough, 17–19  
   index operator, 16–17  
   logical operators, 14–15  
   switch-statements, 17  
 calling a pure virtual function, 212, 220  
 capability queries, 302–306

cast operators vs. conversion operators,  
 24–25  
 casting. *See also* void \*.  
   base class pointers to derived class  
   pointers. *See* downcasting.  
   incomplete types, 100–101  
   under multiple inheritance, 98–100, 147  
   non-dynamic casts. *See* static casts.  
   old-style casts, 102–103  
   to references, 12–13  
   reinterpret\_cast, 76, 100–101  
   static casts, 103–105  
 casting away const, 252  
 casts, maintenance, 76, 84, 102, 103  
 catch clauses, ordering, 184  
 catching  
   exceptions, 182–183  
   string literals, 178–180  
 change logs, 2–3  
 Cheshire Cat technique. *See* Bridge  
   pattern.  
 Clamage, Steve, xv  
 class design  
   aggregation vs. acquaintance, 253–258  
   const data members, 245–247  
   const member functions  
     casting away const, 252  
     meaning of, 250–252  
     semantics and mechanics, 249–250  
     syntax, 248  
   decrement operator, 264–268  
   friend vs. member operators, 262–264  
   get/set interfaces, 241–245  
   increment operator, 264–268  
   operator overloading, 258–264  
   operator precedence, 261–262  
   overloading, 258–262  
   reference data members, 245–247  
   templated copy operations, 268–270

- class hierarchies. *See* hierarchy design.
- class implementation, varying with `#if`, 70–71
- class objects, bitwise copy, 136–138
- classes
  - access protection, 19–23
  - interface, 145
  - POD (Plain Old Data), 136
  - pure virtual base, 24
- cleverness, unnecessary, 29–31
- Cline, Marshall, xvi
- code reuse, 281–285
- coders. *See* programmers.
- coding, conciseness, 2
- Comeau, Greg, xv
- comma (,) operator, 39–40
- Command pattern, 281–285
- comments. *See also* maintenance; readability.
  - avoiding, 2–4
  - change logs, 2–3
  - excessive, 1–4
  - fallthrough, 18
  - maintaining, 1–4
  - self-documenting code, 2
  - specifying ownership, 254–255
- compilation, avoiding recompilation, 21, 202–203
- compiler-generated assignment of virtual base subobjects, 145
- component coupling
  - defeating access protection, 280
  - global variables, 6
  - polymorphism, 202
- Composite pattern, 281, 283–284
- computational constructors, 161
- concrete public base classes, 285–286
- conditional operator (`?:`), 15–16, 40
- const data members, 245–247
- const member functions
  - casting away `const`, 252
  - meaning of, 250–252
  - semantics and mechanics, 249–250
  - syntax, 248
- const objects *vs.* literals, 13–14
- const pointers
  - vs.* array names, 87
  - definition, 50
  - vs.* pointer to `const`, 81
  - pointer-to-`const` conversion, 81–82
- const type-qualifier
  - migrating, 52
  - references, 10–11
- constant-expressions, 67
- constants
  - assigning to, 246
  - vs.* literals, 4–6, 13–14
- `const_cast` operator, 103, 112, 246, 252
- constructors
  - calling virtual functions, 218–220
  - computational, 161
  - conversions, 95–98
  - implementing with template member functions, 268–270
  - initialization *vs.* assignment, 139–141
  - initializing static members, 163–165
  - virtual constructor idiom, 223
- container substitutability, 273–277
- containers of pointers, 255–258
- contravariance, 120–123
- conversion functions, explicit, 90, 136–138
- conversion operators
  - alternative to, 90
  - ambiguous, 90–94
  - diction, 24–25
  - purpose of, 92

- conversions
  - array names *vs.* constant pointers, 87
  - casting
    - incomplete types, 100–101
    - multiple inheritance, 98–100
    - under multiple inheritance, 98–100
    - old-style casts, 102–103
    - `reinterpret_cast`, 76, 100–101
    - static casts, 103–105
  - const pointers
    - vs.* array names, 87
    - pointer-to-const conversion, 81–82
  - `const_cast` operator, 103–105, 112, 246, 252
  - contravariance, 120–123
  - delta arithmetic
    - casting incomplete types, 101
    - class object addresses, 98–100
    - correcting `this` value in virtual function call, 235–236
    - downcasting, 89
  - downcasting, 89
  - `dynami c_cast` operator
    - ambiguity, 116–120
    - to ask a personal question of a type, 300
    - for a capability query, 303–304
    - of pointer to virtual base sub-object, 146
    - in preference to static cast, 89
    - `static_cast` in preference to, 146
  - formal arguments
    - passing by reference, 109
    - passing by value, 108–109
    - temporary initialization, 106–109
  - functions for, 90–94
  - implicit
    - ambiguous results, 90–94
    - constructor conversions, 95–98
    - contravariance, 120–123
    - from derived class to public base, 122
    - initialization of formal arguments, 106–109
    - references, 112–115
  - initializing
    - formal arguments, 106–109
    - references, 112–115
  - Meyers, Scott, 105
  - objects, temporary lifetime, 110–111
  - old-style casts, 102–103
  - platform dependency, 76
  - pointer-to-const conversion, 81–82
  - pointer-to-multidimensional arrays, 87–88
  - pointer-to-pointer-to-base conversion, 86–87
  - pointer-to-pointer-to-const conversion, 82–86
  - pointers
    - converting, 82–86
    - to incomplete class types, 100–101
    - to members, converting, 120–123
    - to pointers to derived classes, 86–87
  - qualification conversions, 82–86
  - references
    - as aliases, 112–115
    - conversions, 112–115
    - to incomplete class types, 100–101
  - `reinterpret_cast`, 76, 100–101
  - slicing derived class objects, 79–81
  - static casts, 103–105
  - temporaries, 110–115. *See also* anonymous temporaries.
  - `void *`, 75–78
  - converting types. *See* casting; `void *`.
  - copy constructor base, initializing, 147–150

copy operations  
   denying, 135  
   idiom, 27–29  
   initialization, 132–136  
   templated, 268–270  
 cosmic hierarchies, 295–299  
 coupling. *See* component coupling.  
 covariant return types, 228  
 cowpath simile for natural language, 26  
 cross-cast, 304  
 cv-qualifiers. *See* const type-qualifier;  
   volatile type-qualifier.

---

## D

dark corners (of the C++ language)  
   address of a non-lvalue return, 267  
   calling a pure virtual function, 212, 220  
   compiler-generated assignment of  
     virtual base subobjects, 145  
   dynamic scope of invocation member  
     operator `delete`, 206  
   `dynamic_cast` to inaccessible base, 177  
   guarantees associated with  
     `reinterpret_cast`, 100  
   ignoring qualifiers on reference type  
     name, 10  
   indexing an integer, 16  
   lvalue result of conditional operator, 16  
   overriding invisible functions, 228  
   point of declaration of an  
     enumerator, 54  
   qualification of function typedef, 52  
   string literal temporary for `throw`  
     expression, 181  
   switch-statement structure, 18

data abstraction  
   for exception types, 178  
   purpose of, 241  
 data hiding. *See* access protection.  
 debug code, in executable modules, 67  
 debugging  
   `#if`, 66–69  
   unreachable code, 67–69  
 declaration-specifiers, ordering, 50–51  
 Decorator pattern, 212, 281  
 decrement operator, 264–268  
 default argument initializer. *See* default  
   initialization.  
 default initialization  
   vs. overloading, 8–9  
   uses for, 7, 9  
   and virtual base objects, 244  
`#define`, and namespace, 62  
`#define` literals, 61–63  
`#define` pseudofunctions, 64–66  
 degenerate hierarchies, 286  
`delete []` operator, 168–170  
   array allocation, 168–170  
   replacing, 173–176  
   scalar allocation, 168–170  
   scope and activation, 176–177  
 delta arithmetic  
   casting incomplete types, 101  
   class object addresses, 98–100  
   correcting `this` value in virtual  
     function call, 235–236  
   downcasting, 89  
 derived class objects, slicing, 79–81  
 derived classes, overriding functions,  
   228–229  
 design firewalls, 202  
 destroyed vs. destructed, 24  
 destructors, calling virtual functions,  
   218–220

developers. *See* programmers.  
 Dewhurst, David, xv  
 diction. *See also* idiom.  
     acronyms, 25–26  
     cast operators, 24–25  
     conversion operators, 24–25  
     destroyed *vs.* destructed, 24  
     function calls, 24  
     member functions, 24  
     methods, 24  
     null pointers, 25  
     pure virtual base classes, 24  
 direct argument initialization, 156–158  
 dominance, 236–239  
 downcasting, 89  
 dynamic binding, 200  
 dynamic scope of invocation member  
     operator `delete`, 206  
`dynamic_cast` operator  
     ambiguity, 116–120  
     to ask a personal question of a type, 300  
     for a capability query, 303–304  
     conversions, 116–120  
     of pointer to virtual base subobject, 146  
     in preference to static cast, 89  
     `static_cast` in preference to, 146  
     virtual base default initialization, 146  
`dynamic_cast` to inaccessible base, 177

---

## E

Eiffel, 29  
*The Elements of Style*, 26  
 enumerators  
     `#define` literals, 63  
     initializing static members, 163  
     magic numbers, 5  
     point of declaration, 54

ethics of programming, 32–33  
 evaluation order. *See* precedence.  
 examples, source code, xiv  
 exception handling, 172–173, 177–184,  
     193–194  
 exception types, 178–180  
 extern types, 55

---

## F

Factory Method pattern, 229, 274–275  
 fallthrough, 17–19  
 for statement  
     variable scope restriction, 45–48  
     *vs.* `while` statement, 47  
 formal arguments  
     passing by reference, 109  
     passing by value, 108–109  
     specifying ownership, 254–255  
     temporary initialization, 106–109  
 forward class declaration. *See* incomplete  
     declaration.  
 free *vs.* `delete`, 168–169  
 freeing  
     arrays and scalars, 167–170  
     resources of heap-allocated objects, 195  
 friend *vs.* member operators, 262–264  
 function matching. *See* overloading.  
 function/object ambiguity, 51  
 function object idiom, 282  
 function typedef, qualification, 52  
 functions  
     binding references to, 13  
     for conversions, 90–94  
     invisible, overriding, 228  
     pointers to, 13  
     references for return values, 11–12  
     references to, 13

---

**G**

Gamma, Erich, xi  
 get/set interfaces, 241–245  
 global variables, 6–8  
 gotchas, definition, xi  
 Gschwind, Thomas, xv  
 guarantees associated with  
     reinterpret\_cast, 100

---

**H**

header files  
     reference counting inclusions, 152–153  
     Schwarz counter, 152–153  
 Hewins, Sarah, xv  
 hiding  
     nonvirtual functions, 209–212  
     vs. overloading and overriding, 224–230  
 hierarchy design  
     address arithmetic errors, 286  
     arrays of class objects, 271–273  
     capability queries, 302–306  
     code reuse, 281–285  
     concrete public base classes, 285–286  
     container substitutability, 273–277  
     cosmic hierarchies, 295–299  
     degenerate hierarchies, 286  
     inheritance, 287–292  
     interface classes, 281  
     protected access, 277–280  
     protocol classes, 281  
     public inheritance, 281–285  
     runtime type queries, 299–301  
     slicing, 286  
     switching on type codes, 292–295  
     type-based control structures, 292–295

value semantics, 286  
 hyphen angle bracket (->) operator,  
     58–60

---

**I**

idiom. *See also* diction.  
     auto\_ptr template, 28–29  
     copy operation, 27–29  
     function object, 282  
     natural language, as a cowpath, 26  
     and natural selection, 27  
     resource acquisition is initialization, 28  
     rules of natural language, 26  
     smart pointer, 59, 282  
     virtual constructor, 223. *See also*  
         Prototype pattern.  
 #i f  
     debugging, 66–69  
     platform independence, 69  
     portability, 69–70  
     in the real world, 71–72  
     varying class implementation, 70–71  
 ignoring qualifiers on reference type  
     name, 10  
 implicit conversions  
     ambiguous results, 90–94  
     constructor conversions, 95–98  
     contravariance, 120–123  
     from derived class to public base, 122  
     initialization of formal arguments,  
         106–109  
     references, 112–115  
 incomplete declaration, 20–21  
 incomplete types  
     casting, 100–101  
     for decoupling, 20–21

increment operator, 264–268  
 index operator, predefined, 16–17  
 index operator ([ ]), 16–17  
 indexing  
   array names, 16  
   integers with pointers, 16–17  
   pointers, 16  
 infix notation, 56–58, 258–259  
 inheritance  
   access protection, 23  
   hierarchy design, 281–285, 287–292  
 initialization  
   vs. assignment, 125–129, 139–141  
   bitwise copy of class objects, 136–138  
   copy constructor base, 147–150  
   copy operations, 132–136  
   default  
     vs. overloading, 8–9  
     uses for, 7, 9  
   direct argument, 156–158  
   direct vs. copy, 153–156  
   formal arguments, temporary, 106–109  
   implicit copy operations  
     bitwise copy of class objects, 136–138  
     description, 132–136  
   initializers, confused with arrays, 35–36  
   member initialization list, ordering,  
     141–142  
   passing arguments, 126  
   references, 112–115  
   return value optimizations, 158–162  
   runtime static, ordering, 150–153  
   scoping variables, 129–132  
   self-initialization, 53–55  
   Singleton pattern, 7–8  
   static members in constructors,  
     163–165  
   virtual base default, 142–147  
 initializers, confused with arrays, 35–36

integers, indexing, 16  
 interface classes, 145, 281, 284–285.  
   *See also* mix-in classes.

---

## J

Josuttis, Nicolai, xvi

---

## K

Kernighan, Brian, xv

---

## L

Lafferty, Debbie, xv  
 language (natural), 26  
 left angle brackets (<<<), Sergeant  
   operator, 48–49  
 lexical analysis, 49  
 literals  
   vs. `const` objects, 13–14  
   vs. constants, 4–6  
   defining, 61–63  
 local addresses  
   disappearing stack frames, 185  
   idiomatic problems, 186–187  
   static interference, 186  
 local scope problems, 187  
 local variable lifetimes, 187  
 logical operators, 14–15, 40  
 lvalue  
   binding references, 11, 112–115  
   definition, 13–14  
   function return, 11

initializing references. *See* binding,  
 reference to lvalue.  
 nonmodifiable, 14, 62  
 result of conditional operator, 15–16

---

## M

macros, side effects, 16, 64–66  
 magic numbers, 4–6  
 maintenance. *See also* comments;  
 readability.  
 and casts, 76, 84, 102, 103  
 easing  
   assertions, 73  
   coding standards, 32, 40–41  
   container ownership, 255  
   container substitutability, 275  
   declaration-specifier ordering, 51  
   fallthrough, 17–18  
   idioms, 27  
   incomplete declarations, 20  
   initializing static members, 164  
   mnemonic names, 3  
   naming conventions, 3  
   precedence, 43–44  
   scalar allocation *vs.* array, 170  
   for statement, 45, 48  
   type codes, 202  
   typed-base control structures,  
   293–294  
 made difficult  
   asking personal questions of  
   objects, 299  
   auto\_ptr, 196  
   code reuse, 281, 285  
   comments, 1–4  
   const and reference data  
   members, 245  
   dynamic\_cast, 116  
   global variables, 6–8  
   implicit conversions, 92  
   memory allocation failure, 171  
   miscommunication of container  
   ownership, 256  
   naming conventions, 23  
   overloading virtual functions, 215  
   public inheritance, 281, 285  
   resource acquisition is initialization,  
   193–194  
   switch on type codes, 200  
   throwing/catching string literals, 178  
   type-based control structures, 293  
   unnecessary cleverness, 29–31, 115  
 made impossible  
   capability queries, 304  
   cosmic hierarchies, 296–297  
   *#if*, 69, 71  
   initializing static members, 165  
   local address abuse, 185  
   platform dependence, 69  
   static interference, 186  
   varying class implementation, 71  
   remote changes (bugs caused by), 77,  
   101, 103, 105  
 malloc *vs.* new, 168–169  
 maximal munch  
   description, 48–49  
   examples, 30, 48–49  
 McKillen, Patrick, xv  
 meaningful names. *See* naming  
   conventions.  
 member functions  
   diction, 24  
   template, 29  
   virtual static, 205–206  
 member initialization list, ordering, 29,  
 141–142

member vs. friend operators, 262–264

members

- pointers to, 9, 120–123
- requiring initialization, 139

memory and resource management

- allocation failure, 171–173
- `auto_ptr`, 195–197
- catch clauses, ordering, 184
- catching exceptions, 182–183
- catching string literals, 178–180
- exception handling, 177–184
- exception types, 178–180
- freeing
  - arrays and scalars, 167–170
  - resources of heap-allocated objects, 195
- local addresses
  - disappearing stack frames, 185
  - idiomatic problems, 186–187
  - static interference, 186
- local scope problems, 187
- local variable lifetimes, 187
- memory leaks, 187, 253
- replacing global `new` and `delete`, 173–176
- resource acquisition is initialization, 190–195
- scalar vs. array allocation, 167–170
- scope and activation, `new` and `delete`, 176–177
- static fix, 188–190
- throwing anonymous temporaries, 181–182
- throwing pointers, 181
- throwing string literals, 177–180

memory leaks, 187, 253–258

methods, 24

Meyers, Scott, xvi, 105

migrating type-qualifiers, 52

mix-in classes, 281. *See also* interface classes.

mnemonic names, 3

Monostate pattern, 203–204

multiple inheritance, casting, 98–100

---

## N

named return value optimization (NRV), 161

namespace

- anonymous, 55
- and `#define`, 62

naming conventions

- access protection, 23
- mnemonic names, 3
- self-documenting code, 3
- simplicity, 23
- specifying ownership, 254–255
- variable type in variable name, 23

NDEBUG, mysterious failures, 67

nonvirtual base class destructor

- addresses of base class subobjects, 206–208
- exceptions, 208–209
- undefined behavior, 205
- virtual static member functions, 205–206

NRV (named return value optimization), 161

null

- `dynami c_cast` result, 117
- pointers, 25
- references, 11

Null Object pattern, 282, 284, 294

numeric literals vs. constants, 4–6

---

**O**

object-oriented to non-object-oriented communication, 202

objects, temporary lifetime, 110–111

old-style casts, 102–103, 278

Oldham, Jeffrey, xv

operator delete, 206

operator new

- allocation failure, 172
- replacing, 173–176
- scalar allocation, 168–170
- scope and activation, 176–177

operator overloading, 258–264

operator precedence, 261–262

operators

- , (comma operator), 39–40
- ?: (conditional operator), 40
- > (arrow operator), 58–60
- && (logical operator), 40
- || (logical operator), 40
- <<< (Sergeant operator), 48–49

C++ base language, 14–17

cast, 24–25

cast vs. conversion, 24–25

conversion, 24–25, 90–94

evaluation order, 39–41

function lookup, 56–58

index operator, predefined, 16–17

logical, 14–15

lvalue, result of conditional operator, 15–16

new [] operator, 39

operator function lookup, 56–58

overloading

- > (arrow operator), 58–60
- evaluation order, 41
- operator function lookup, 56–58
- operators, 56–58
- precedence, 41

- precedence, 17, 39–40
- predefined index operator, 16–17

overloading

- > (arrow operator), 58–60
- ambiguities, 5
- in class design, 258–262
- vs. default initialization, 8–9
- vs. hiding and overriding, 224–230

increment/decrement operators, 264

infix notation, 56–58

operators, 56–58, 258–264

virtual functions, 214–215

overriding

- definition, 232
- invisible functions, 228
- mechanisms, 230–236
- vs. overloading and hiding, 224–230

ownership. *See* aggregation vs. acquaintance.

---

**P**

parallel hierarchies, 274–275

parentheses ( ( ) ), allocating arrays, 35

passing arguments, 126

patterns

- Acyclic Visitor, 306
- Bridge, 21–22
- Command, 281–285
- Composite, 281, 283–284
- Decorator, 212, 281
- Factory Method, 229, 274–275
- Monostate, 203–204
- Null Object, 282, 284, 294
- Prototype, 223, 229, 282–283
- Proxy, 294
- Singleton, 7–8, 152, 204
- Strategy, 291–292

- Template Method, 212–214
  - Visitor, 215, 227, 306
- pencil cup, battleship in, 295
- personal questions (about an object's type), 116, 200, 275, 299–301
- pimpl idiom. *See* Bridge pattern.
- placement new
  - evaluation order of arguments, 39
  - invoking constructor, 127, 138, 155
  - replacing global new and delete, 174
- platform dependence
  - conversions, 76
  - literals *vs.* constants, 5
- POD (Plain Old Data) classes, 136
- point of declaration of an enumerator, 54
- pointer formal arguments, 49
- pointer-to-const conversion, 81–82
- pointer-to-multidimensional array, 87–88
- pointer-to-pointer-to-base conversion, 86–87
- pointer-to-pointer-to-const conversion, 82–86
- pointers
  - containers of, 255–258
  - converting, 82–86
  - to functions, 13
  - to incomplete class types, 100–101
  - to local variables, 185
  - to members, 9
  - to members, converting, 120–123
  - ownership, 255–258
  - to pointers to derived classes, 86–87
  - precedence problems, 43–44
  - vs.* references, 10–13
  - throwing, 181
- polymorphism
  - algorithms, variant and invariant, 212–214
- component coupling, 202
- design firewalls, 202
- dominance, 236–239
- dynamic binding, 200
- flexibility of template methods, 212–214
- hiding
  - nonvirtual functions, 209–212
  - vs.* overloading and overriding, 224–230
- nonvirtual base class destructor
  - addresses of base class subobjects, 206–208
  - exceptions, 208–209
  - undefined behavior, 205
  - virtual static member functions, 205–206
- object-oriented to non-object-oriented communication, 202
- overloading
  - vs.* hiding and overriding, 224–230
  - virtual functions, 214–215. *See also* default initialization.
- overriding
  - definition, 232
  - mechanism, 230–236
  - vs.* overloading and hiding, 224–230
- switching on type codes, 200
- type codes, 199–204
- virtual assignment, 220–224
- virtual copy construction, 223
- virtual functions
  - calling in a nonvirtual manner, 211
  - calling in constructors and destructors, 218–220
  - default argument initializers, 216–217. *See also* overloading, virtual functions.
  - overloading, 214–215. *See also* default initialization.

portability  
  *#if*, 69–70  
  null pointers, 25

precedence  
  , (comma operator), 39–40  
  ?: (conditional operator), 40  
  && (logical operator), 40  
  || (logical operator), 40  
  and associativity, 42, 44–45  
  fixing, 39–41  
  index operators, 17  
  levels of precedence, 42  
  new [] operator, 39  
  operator overloading, 41  
  operators, 39–40, 261–262  
  overview, 36–37  
  pointers, 43–44

predefined index operator, 16–17

preprocessor  
  *assert* macro, 72–74  
  assertions, side effects, 72–74  
  class implementation, varying with *#if*,  
  70–71  
  constant-expressions, 67  
  debug code, in executable modules, 67  
  debugging, 66–69  
  *#define* literals, 61–63  
  *#define* pseudofunctions, 64–66  
  *#if*  
  debugging, 66–69  
  platform independence, 69  
  portability, 69–70  
  in the real world, 71–72  
  varying class implementation, 70–71  
  literals, defining, 61–63  
  NDEBUG, mysterious failures, 67  
  pseudofunctions, defining, 64–66  
  scope, *#define* literals, 61–63  
  preprocessor macros, side effects, 16

programmers  
  adolescent behavior, 31–33  
  ethical duties, 32–33  
  unnecessary cleverness, 29–31

*Programming in C++*, 3

protected access, 277–280

protocol classes, 281. *See also* interface  
  classes.

Prototype pattern, 223, 229, 282–283

Proxy pattern, 294

pseudofunctions, defining, 64–66

public inheritance, 281–285

pure virtual base classes, 24

pure virtual functions, calling, 212

---

## Q

qualification conversions, 82–86

qualification of function typedef, 52

question mark colon (:?) conditional  
  operator, 15–16, 40

---

## R

readability. *See also* comments;  
  maintenance.  
  formatting code, 29–31  
  unnecessary cleverness, 29–31

recompilation, avoiding, 21, 283

reference counting, 257

reference counting inclusions, 152–153.  
  *See also* Schwarz counter.

reference data members, 245–247

reference type name, ignoring qualifiers, 10

references  
  as aliases, 10–13, 112–115

- to arrays, 12
- binding to functions, 13
- binding to lvalue, 11
- casting objects, 12–13
- const type-qualifier, 10–11
- conversions, 112–115
- to functions, 13
- to incomplete class types, 100–101
- initializing, 112–115
- to local variables, 185
- null, 11
- vs.* pointers, 10–13
- return values for functions, 11–12
- underusing, 10–13
- volatile* qualifiers, 10–11
- `reinterpret_cast`, 76, 100–101, 146
- remote changes (bugs caused by), 77, 101, 103, 105
- resource acquisition is initialization, 28, 190–195
- resource handle. *See* resource acquisition is initialization.
- resource management. *See* memory and resource management.
- resources, freeing heap-allocated objects, 195
- return value optimization (RVO), 158–162
- reuse
  - code, 281–285
  - global variables, 6
  - white-box. *See* inheritance.
- runtime static initialization, ordering, 150–153
- runtime type queries, 299–301
- RVO (return value optimization), 158–162

---

## S

- Saks, Dan, xv, xvi
- scalars, freeing, 167–170
- Schwarz, Jerry, 152
- Schwarz counter, 152–153. *See also* reference counting inclusions.
- scope
  - `#define` literals, 61–63
  - local scope problems, 187
  - restriction, variables, 45–48
- scoping variables, initialization, 129–132
- self-documenting code
  - avoiding comments, 2
  - naming conventions, 3
- self-initialization, 53–55
- Semantics, xiv
- Sergeant operator (`<<<`), 48–49
- set/get interfaces, 241–245
- Singleton pattern, 7–8, 152, 204
- slicing
  - derived class objects, 79–81
  - hierarchy design, 286
- smart pointer idiom, 59, 282
- social commentary
  - adolescent behavior, 31–33
  - array/initializer confusion, 36
  - capability queries, 302
  - increment/decrement operators, 267
  - old-style casts, 102
  - operator overloading, 258
  - personal questions of an object, 299–300
  - unnecessary cleverness, 29
- Software Engineering Code of Ethics...*, 32
- square brackets (`[ ]`), allocating arrays, 35

- Stark, Kathy, 3
- static casts, 103–105, 278
- static members in constructors,
  - initializing, 163–165
- static types, 55
- static variables, runtime static initialization problems, 151
- `static_cast`, 103–105, 278
- Strategy pattern, 291–292
- string literal temporary for throw expression, 181
- string literals, throwing, 177–181
- Stroustrup, Bjarne, 295
- Strunk, William, 26
- subexpressions, evaluation order, 37–39
- Sutter, Herb, xvi
- switch-statement structure, 18
- switch-statements, 17
- switching on type codes, 200, 292–295
- syntax
  - arrays
    - confused with initializers, 35–36
    - migrating type-qualifiers, 52
  - associativity
    - and precedence, 42
    - problems, 44–45
  - const member functions, 248
  - const pointers, 50
  - const type-qualifier, migrating, 52
  - declaration-specifiers, ordering, 50–51
  - evaluation order
    - , (comma operator), 39–40
    - ?: (conditional operator), 40
    - && (logical operator), 40
    - || (logical operator), 40
    - fixing, 39–41
    - new operator, 39
    - operator overloading, 41
    - overview, 36–37
    - placement new, 39
    - subexpressions, 37–39
  - extern types, 55
  - function/object ambiguity, 51
  - infix notation, 56–58
  - initialization
    - initializers, confused with arrays, 35–36
    - self-initialization, 53–55
  - initializers, confused with arrays, 35–36
  - lexical analysis, 49
  - maximal munch, 48–49
  - migrating type-qualifiers, 52
  - new [] operator, 39
  - operator function lookup, 56–58
  - operator overloading
    - evaluation order, 41
    - operator function lookup, 56–58
  - overloading
    - > (arrow operator), 58–60
    - infix notation, 56–58
    - operators, 56–58
  - placement new, evaluation order, 39
  - pointer formal arguments, 49
  - pointers, precedence problems, 43–44
  - precedence problems
    - and associativity, 42, 44–45
    - levels of precedence, 42
    - pointers, 43–44
  - scope restriction, variables, 45–48
  - self-initialization, 53–55
  - for statement
    - variable scope restriction, 45–48
    - vs. `while` statement, 47
  - static types, 55
  - subexpressions, evaluation order, 37–39
  - templates, instantiating, 49

token identification, 48–49  
 type-qualifiers, migrating, 52  
 types, linkage-specifiers, 55  
 variables, scope restriction, 45–48  
 volatile type-qualifiers,  
   migrating, 52  
 while statement vs. for statement, 47

---

## T

Template Method pattern, 212–214  
 template methods, flexibility, 212–214  
 templated copy operations, 268–270  
 templates, instantiating, 49  
 temporaries, conversions, 112–115  
 temporary objects, 110–111  
 terminology. *See* diction.  
 throw expression, 181  
 throwing  
   anonymous temporaries, 181–182  
   pointers, 181  
   string literals, 177–180  
 thunks, 235  
 token identification, 48–49  
 type-based control structures, 292–295  
 type codes, 199–204, 292–295  
 type-qualifiers  
   const  
     migrating, 52  
     references, 10–11  
   volatile  
     migrating, 52  
     references, 10–11  
 types  
   converting. *See* casting; void \*.  
   linkage-specifiers, 55

---

## U

unnecessary cleverness, 29–31  
 uses. *See* acquaintance.

---

## V

value semantics, 286  
 variables  
   encoding type in name, 23  
   scope restriction, 45–48  
 vectors vs. arrays, 36, 168, 196  
 vertical lines (||) logical operator, 40  
 virtual assignment, 220–224  
 virtual base default, initializing,  
   142–147  
 virtual constructor idiom, 223. *See also*  
   Prototype pattern.  
 virtual copy construction, 223  
 virtual functions  
   calling in a nonvirtual manner, 211  
   calling in constructors and destructors,  
     218–220  
   default argument initializers,  
     216–217  
   overloading, 214–215  
   pure, calling, 212  
 virtual static member functions,  
   205–206  
 visibility vs. access protection, 19–23  
 Visitor pattern, 215, 227, 306  
 void \*, 75–78. *See also* casting.  
 volatile type-qualifier  
   migrating, 52  
   references, 10–11  
 vptr (pointer to a vtbl), 231  
 vtbl (virtual function table), 231–236

---

## W

while statement *vs.* for statement, 47

White, E.B., 26

white-box reuse. *See* inheritance.

Wilson, Matthew, xv

word choice. *See* diction.

*Writings from The New Yorker*, 26

---

## Z

Zolman, Leor, xv