INDEX

Symbols
< > (angle brackets), in templates, 383
|| operator, 21
# character, 5
! operator, 13, 21
!= operator, 20
% operator, 13
%= operator, 13
& operator, 14
&& operator, 21
&= operator, 14
* operator, 13
*= operator, 13
+ operator, 13
+= operator, 13
- operator, 13
-= operator, 13
/ operator, 13
/= operator, 13
<< operator, 14
output streams and, 412
overloading, 423–425
<<< operator, 14
<= operator, 20
= operator, 13
== operator, 20
> operator, 20
>> operator, 14
input streams, 417–418
overloading, 423–425
>>= operator, 14
>= operator, 20
^ operator, 14
^= operator, 14
| operator, 14
|= operator, 14

A
abs() function, 511, 695
abstract classes, 303
abstraction
design, 141
implementation, 138–139
interface, 138–139
reusable code and, 145–146
access, 201
accumulate algorithm, 611–612
adaptor function objects
binders, 624–626
member functions, call, calling, 628–629
negators, 626–627
adaptor patterns, 1006–1008
adjacent_find() algorithm, 524
Agile Manifesto, 866–867
<algorithm> header, 630–631
algorithms, 630–631
accumulate, 611–612
auditing voter registration example,
657–661
binary search, 649–650
binary_search(), 529
equal_range(), 529
lower_bound(), 529
upper_bound(), 529
callbacks, 607
comparison, 524–525
equal(), 525
lexicographical_compare(), 525
mismatch(), 525
counting, 525
all_of(), 525
any_of(), 525
count(), 525
count_if(), 525
none_of(), 525
find(), 608–611
find_all(), 735–736
find_if(), 608–611
header files, 1078–1079
heap
is_heap, 530
is_heap_until(), 530
make_heap(), 530
pop_heap(), 530
push_heap(), 530
sort_heap(), 530
iterators, 521–522, 608, 631, 737
minimum-maximum, 653–654
clamp(), 530
max(), 530
max_element(), 530
min(), 530
min_element(), 530
minmax(), 530
minmax_element(), 530
modifying sequence
copy(), 525, 638–639
copy_backward(), 525
copy_if(), 525
copy_n(), 525
fill(), 526
fill_n(), 526
generate(), 526
generate_n(), 526
move(), 526, 640–641
move_backward(), 526
random_shuffle(), 526
remove(), 526, 641–643
remove_copy(), 526
remove_copy_if(), 526
replace(), 526, 641
replace_copy(), 526
replace_copy_if(), 526
replace_if(), 526
reverse(), 526, 644
reverse_copy(), 526
sample(), 526, 643–644
shuffle(), 526, 644
transform(), 526, 637–638
unique(), 526, 643
unique_copy(), 526
move semantics, 612
non-modifying, 523–524
numerical processing, 655
accumulate(), 531
adjacent_difference(), 531
exclusive_scan(), 531
gcd(), 530, 656
inclusive_scan(), 531
inner_product(), 656
inner_product(), 531
iota(), 530, 656
lcm(), 531, 656
partial_sum(), 531
reduce(), 531, 656
transform_exclusive_scan(), 531
transform_inclusive_scan(), 531
transform_reduce(), 531, 657
operational
for_each(), 527, 644–646
for_each_n(), 527, 646
parallel, 655
partition, 647–648
is_partitioned(), 527
partition(), 527
partition_copy(), 528
partition_point(), 528
stable_partition(), 528
permutation
  is-permutation( ), 532
  next-permutation( ), 532
  prev-permutation( ), 532
resources, 1068
scan algorithms, 657
search, 524
  adjacent_find( ), 524
  find( ), 524
  find_end( ), 524
  find_first_if( ), 524
  find_if( ), 524
  find_if_not( ), 524
  search( ), 524
  search_n( ), 524
sequence algorithms
  modifying, 636–644
  non-modifying, 631–636
set, 650–653
  includes( ), 529
  inplace_merge( ), 529
  merge( ), 529
  set_difference( ), 529
  set_intersection( ), 529
  set_symmetric_difference( ), 529
  set_union( ), 529
sorting, 649
  is_sorted( ), 528
  is_sorted_until( ), 528
  nth_element( ), 528
  partial_sort( ), 528
  partial_sort_copy( ), 528
  sort( ), 528
  stable_sort( ), 528
Standard Library, 507, 521
swap and exchange
  exchange( ), 527, 647
  iter_swap( ), 527
  swap( ), 527, 646
  swap_ranges( ), 527
writing, 735–737
aliases, 334
alias templates, 399

namespaces, 10
aliasing, 183, 189
allocate( ) method, 728–729
allocating arrays, dynamically, 31–32
allocating memory, 166–168
Allocator type, 728–729
allocators, header files, 1078–1079
all_of( ) algorithm, 525, 636
anonymous namespaces, 349
<any> header, 713–714
any_of( ) algorithm, 525, 636
APIs (application programming interface), 140
architecture, crossplatform development and
  address sizes, 1020
  binary compatibility, 1019–1020
  byte order, 1020–1021
  integer size, 1018–1019
arguments
  default, 257–258
  types, operator overload, 476
arithmetic function objects, 621
arithmetic operators
  durations and, 695
  overloading, 269–270
  decrement, 483–484
  increment, 483–484
  unary minus, 483
  unary plus, 483
array container, 516, 520, 568–569
arrays
  allocating, dynamically, 31–32
  associative arrays, 518
  basic types, 168–170
  constant expressions and, 23
  declaring, heap, 169
deleting, 171
dynamic, 169
initializing, 23
multi-dimensional
  heap, 173–175
  stack, 172–173
of objects, 170
one-dimensional, 24
as pointers, 177–178
std::array container, 25
std::vector container, 25–26
ASCII (American Standard Code for
Information Interchange), 663–664
assertions, debugging and, 945–946
assign( ) method, 545
assignment operators, 236–242
copy, 228
declaring, 225–226
defaulted, explicit, 227
defining, 226–227
deleted, explicit, 227
move, 243, 245–246
swap( ) function, 247–248
objects, as return values, 228–229
associative arrays, 518
at( ) function, 376
atan2( ) function, 511
<atomic> header, 827
atomic operations, 830–831
atomic types, 827–830
attributes
[[deprecated]], 364
[[fallthrough]], 364
[[maybe_unused]], 365
[[nondiscard]], 364–365
[[noreturn]], 363
vendor-specific, 365
auto keyword, 40, 613
automatic variables, 164
auto_ptr, 192

B
bad( ) method, 414–415
base( ) method, 731
base classes, 130, 278, 302–304
    ambiguous, 311–312
    virtual, 312, 331–332
begin( ) function, 523
behaviors, 126
Bessel functions, 511
beta functions, 511
bidirectional streams, 431–432
big-endian ordering of bytes, 1021
binary logical operators, overloading, 484–485
binary operators, 13
binary_search( ) algorithm, 649
bind( ) negator adaptor, 624–626
binding
dynamic, 287
early, 287
late, 287
method hiding and, 287
static, 287
bitset container, 519, 521, 600–601
bitwise copying, 236
bitwise function objects, 624
bitwise operators, overloading, 484–485
black-box testing, 909
Boehm, Barry W., 863
boolalpha input manipulator, 423
boolalpha output manipulator, 415
boyer_moore_horspool_searcher algorithm, 633
boyer_moore_searcher algorithm, 633
bridge pattern, 272
buffered streams, 410, 413
bugs. See also debugging
    avoiding bugs, 934–935
    Bugzilla, 912
    life cycle, 910–911
    root causes, 934
    tracking, 912–913
byte order, 1020–1021

C
C++ resources, 1063–1064
C++11, resources, 1067
C++14, resources, 1067
C++17, resources, 1067
cache lines, 818
callbacks, 607, 610
calling member functions, 628–629
casting, 299–300
casting variables, 12
pointers and, 176
casts
const_cast( ), 357–358, 361
dynamic_cast( ), 360–361
reinterpret_cast( ), 359–361
static_cast( ), 358–359, 361
catch exceptions, object handling and, 886
catching exceptions, 437–439, 976
in class hierarchy, 448–449
const reference, 440–441
multiple, 441–443
cbegin( ) function, 523
ceil( ) function, 511, 695
cend( ) function, 523
cerr stream, 410
chain of responsibility pattern, 1010–1014
chaining constructors, 295
character classification, localization and,
669–670
character sets, 664
non-Western, 665
character types, 665
Chrono library
clocks, 698–700
durations, 694–698
time points, 700–702
cin stream, 410
class constants, 260–261
class templates, 514
< > (angle brackets), 383
alias templates, 399
compiler and, 383–384
instantiation, 384
types and, 384
definitions
in header files, 384–385
in source files, 385–386
driving from, 397–398
inheritance, 397–398
instantiation, 378, 382
parameters, 386
deduction for constructors, 389–391
default values, 389
non-type parameters, 387–388
writing
coding without templates, 375–378
Grid class, 378–381
Grid template, 382–383
classes, 124–125
abstract, 303
access specifiers, 201-202
base, 278, 302–304
ambiguous, 311–312
virtual, 312
constructors, 201
data members, 200, 201
declarations, ordering, 203
defining
constructor initializer, 41–42
constructors, 41–42
data members, 40
destructors, 41–42
methods, 40
definitions, 200–201
Derived, 279–281
derived, 304–305
functionality, 292–294
parent constructors, 294–296
parent destructors, 296–297
deriving from existing, 974
destructors, 201
exceptions, writing, 449–452
friends, 232–233
Grid, 378–383
hash tables, 132
hierarchies, catching exceptions, 448–449
high_resolution_clock, 698–699
Impl, 273
implement class, 272–275
instances, 125, 201
interface class, 272–275
interviews, 1046–1050
istream_iterator, 730
Logger, 938
member functions, 201
member variables, 201
members, 201, 203
methods, 201
mixin, 138
nested, 263–264
opaque, 146
ostream_iterator, 729
pair, 576–577
parent, 278
reference_wrapper, 537
rule of zero, 250
statements, 201
std::ostringstream, 425
string, 509
string_view, 509
superclasses, 278
system_clock, 698
template classes, 977–979
time_point, 698, 700–702
types, enumerated, 264–265
using, 43
writing, 972–974
cleanup( ) function, 248
clocks, 698–700
  high_resolution_clock class, 698–699
  system_clock class, 698
  time_point class, 698, 700–702
clog stream, 410
clone( ) method, 375
code
  compiled, 5
decomposition, 81–83
reuse
  advantages, 105
  APIs, 104
  big-O notation, 108–109
  capabilities, 107–108
design, 146–148
design-by-contract, 150
disadvantages, 105–106
error checking, 150
extensibility and, 151–153
frameworks, 104
libraries, 104
licensing and, 110–111
open-source libraries, 112–114
performance and, 109–110
platforms and, 110
versus polymorphism, 131–132
postconditions, 150
preconditions, 150
prototypes, 111–112
resources, 111
safe code, 150–151
SRP (Single Responsibility Principle),
  146–147
stand-alone classes, 104
stand-alone functions, 104
Standard Library, 114
support and, 110–111
templates, 148–150
third-party apps, 112
types of code, 104
writing reusable, 102–103
code points, character sets, 665
coding style, 71. See also design
challenges, 90–91
constants, 87
documentation, comments, 72–81
exceptions, custom, 88
formatting
curly braces ({ }, 88–89
parentheses, 89–90
spaces, 89–90
tabs, 90
interviews, 1041–1042
planning, 72
references, 87
collections, 126
collisions, 591
comments, 3–4
ad hoc, 80–81
complicated code and, 74–75
every line, 77–78
fixed-format, 79–80
meta-information and, 75–76
prefixes, 78
self-documenting code, 81
usage explanation, 72–74
comparison algorithms, 524–525, 634–635
comparison function objects, 622–623
comparison operators, overloading, 270–271
compiled code, 5
compiler-generated constructors, 222–223
component interface, 140
components, 125
computer architecture, resources, 1071
concatenation, tuples, 71
condition variables, 840–842
conditional statements
if/else, 17–18
operator, 20
switch, 18–19
<condition_variable> header, 840
c console, streams and, 410
const keyword
constants, 35
methods, 345
parameter protection, 35
parameters, 343
pointers, 343–345
references, 37, 345
variables, 343
const methods, 251–252
const reference data members, 262–263
const static data members, 260–261
const value, 334–335
constant iterators, 631
constants
class constants, 260–261
null pointer, 32–33
const_cast( ), 357–358, 361, 376
constexpr keyword
constant expression, 346
literal type, 346
constructors, 209
chaining, 295
compiler-generated, 222–223, 228
converting constructors, 307
copy, 218–219
default, 212–215
compiler-generated, 213–214
explicitly defaulted, 214
explicitly deleted, 215
need, 212
writing, 212–213
delegating, 211, 222, 238–239
derived classes, 294–296
error handling, 462–464
function-try-blocks, 464–467
on the heap, 210
inherited, 316–319
initializer-list constructors, 220–222
initializers, 215–218
move, 245–248
multiple, 211
overloading, 211
parent, 294–296
on the stack, 210
typed constructors, 307
writing, 209–210
container adapters
priority_queue, 572–575
queue, 570–572
stack, 575–576
containers
Allocator type, 728–729
allocators, 760
associative, 536, 760–773
bitset, 519, 600–604
container adapters, 536
error checking, 539
hash functions, 738–740
hash map interface
assignment operators, 745–746
deleting elements, 744
hash template parameter, 741
implementation, 741–742
inserting elements, 744
KeyEqual template parameter, 741
removing elements, 745
searching elements, 742–744
swapping elements, 745
type aliases, 741
hash maps
  buckets, 738
  hash functions, 738–740, 746–747
  methods, 748–750
  type aliases, 747–748
hash tables, 536, 591–598
header files, 1077–1078
interfaces, 514
iterators, 539–541, 750–751
  access methods, 758–759
  const_hash_map_iterator class, 751–752
  const_hash_map_iterator method implementations, 752–756
  hash_map_iterator method implementations, 756–757
  type aliases, 758–759
  using, 759–760
ordered associative, 576
reversible, 760
sequential, 536, 737, 773
  array, 568–569
deque, 562
  forwards_list, 566–568
  list, 562–566
standard C-style arrays, 598–599
Standard Library, 507, 514
  array, 516, 520
  bitset, 521
deque, 516, 520
  forward_list, 515, 520
  list, 515, 520
map, 518
  map multimap, 521
  multimap, 518
  multiset, 517–518
  priority_queue, 516–517, 520
queue, 516, 520
set, 517–518
  set multiset, 521
stack, 517, 521
unordered_map, 521
unordered_multimap, 521
unordered_multiset, 521
unordered_set, 521
vector, 342, 514–515, 519
strings, 599–600
unordered associative containers, 518–519, 536, 591–598
writing, 737–773
conversions
  implicit, 267
localization, 667–668
converting constructors, 307
cooked mode literal operators, 366
copy( ) algorithm, 525
copy( ) method, 732
copy assignment operator, compiler generated, 228
copy constructors, 218–219, 236–242
copy list initialization, 45
copy-and-swap idiom, 975–976
copy_backward( ) algorithm, 525
copy_if( ) algorithm, 525
copy_n( ) algorithm, 525
core dumps, 946–947
count( ) algorithm, 525, 636
count_if( ) algorithm, 525
counting algorithms, 525, 636
cout stream, 410
  standard output, 412
  crash dumps, 946–947
  crbegin( ) function, 523
createDocArray( ) function, 169
cref( ) function, 537
crend( ) function, 523
cross-language development
  C and C++, 1024
C code, 1028–1030
C++ code
  calling assembly code from, 1036–1037
  calling from C#, 1030–1031
  calling from Java with JNI, 1031–1033
  calling from scripts, 1034–1036
  calling scripts from, 1033–1034
facade, 1024
object-oriented wrapper, 1024–1025
cross-platform development
architecture and
  address sizes, 1020
binary compatibility, 1019–1020
byte order, 1020–1021
integer size, 1018–1019
cross-compiling, 1020
implementation
  compilers and, 1021–1022
  library implementations, 1022
platform-specific features, 1022–1023
<cstdint> header, 514
C-style arrays, 598–599
cylindrical Neumann functions, 511

dangling pointers, 197, 238
data members, 200
  moving, 247
  mutable, 253
  pointers, type aliases, 355–356
  references, 261–263, 336
  static, 258
    accessing, 259–260
    inline variables, 259
  data races, 815–817
  data structures, resources, 1068
deadlocks, multithreaded programming and, 817–818
deallocate( ) method, 728
deallocating memory, 166–168
debugging
  article citations example, 957–969
  assertions, 945–948
  avoiding bugs, 934–935
  catastrophic bugs, 934
  cosmetic bugs, 934
  crash dumps, 946–947
debug traces
  debug mode, 937–941
  ring buffers, 942–945
  error logging and, 935–937
interviews, 1060
memory problems, 951, 954–956
  memory-access errors, 953–954
  memory-freeing errors, 952–953
multithreaded programs, 956–957
noncatastrophic bugs, 934
regressions, 951
reproducing bugs, 948–949
  debugging nonreproducible, 950–951
  debugging reproducible, 949
resources, 1072
root causes of bugs, 934
tests, 927
dec input manipulator, 423
dec output manipulator, 415
declaring functions, 5, 21–22
decltype keyword, 40
decomposing code, 21
decomposition, 81–82
  modular, 83
  refactoring
    abstraction, 82
    breaking code apart, 82
    locations, 82
    names, 82
decorator pattern, 1008–1010
default arguments, 257–258
default keyword, 214–215
default_searcher algorithm, 633
#define directive, 6
delegating constructors, 211, 222, 238–239
delete[ ], 169
delete( ) operator, 32
delete keyword, deallocating memory, 166–167
delete operator, 500–506
[[deprecated]] attribute, 364
dequ container, 516, 520, 562
dereferencing operators, overloading, 492–494
  operator- >*, 495–496
  operator *, 494
  operator -, 494
dereferencing pointers, 30, 175–176
Derived class, 279–281
derived classes, 304–305
  constructors, parent, 294–296
derived constructor – dynamic strings

destructors, parent, 296–297
functionality, 292–294
Derived constructor, 294–296
design. See also reusable code
abstraction and
benefits, 100
incorporating, 101
C++, 99–100
chess program, 114–115
algorithms, 118–120
class hierarchies, 118
classes, 118–120
data structures, 118–120
error handling, 120–121
patterns, 118–120
subsystems, 115–117
threading models, 117
frameworks and, interviews, 1060
functional requirements and, 96
importance of, 97–99
multithreaded programming, 853–855
non-functional requirements and, 96
object-oriented design, interviews,
1044–1045
patterns, 99, 103, 991
adaptor pattern, 1006–1008
chain of responsibility, 1010–1014
decorator pattern, 1008–1010
dependency injection, 993
factory patterns, 997–1004
interviews, 1060–1061
iterator pattern, 992–993
observer pattern, 1014–1016
proxy pattern, 1004–1006
resources, 1072
singleton, 993–997
program design, interviews, 1042–1044
software design, 96
stakeholders and, 96
Standard Library and, 114
techniques, 103, 971–972
design-level efficiency, 889
caching, 889–890
object pools, 890–891, 893–894
implementation, 891–892
destruction, variables, non-local, 351
destructors, 224–225, 235–236
derived classes, 296–297
error handling and, 467
parent, 296–297
virtual keyword, 288–290
DIP (Dependency Inversion Principle), 159
direct list initialization, 45
directives
endif, 367–368
ifndef, 367–368
#include, 368
#pragma once, 368
preprocessor, 6
directories, iteration, 722–723
DLLs (Dynamic Link Library), 354–355
documentation
comments
ad hoc, 80–81
complicated code and, 74–75
every line, 77–78
fixed-format, 79–80
meta-information, 75–76
prefixes, 78
self-documenting code, 81
usage explanation, 72–74
man pages, 156
doSomething( ) function, 453
double dispatch, 981–985
double-ended queue, 516
doubleInts( ), 178
doubly linked lists, 515
do/while loop, 27
downcasting, 300
doWorkInThread( ) function, 825
dumpRange( ) function, 652
durations, 694–698
dynamic arrays, 169
dynamic binding, 287
dynamic casts, pointers and, 176
dynamic memory, 164–165
allocation, 233–235
dynamic strings, 58
C-style
copyString( ) function, 58
dynamically allocating arrays – exceptions

```plaintext
sizeof( ) operator, 59
strlen( ) function, 58–59
nonstandard strings, 69
std::string class, 62–67
std::string_view class, 67–69
string literals
  assigning, 60
  literal pooling, 60
  raw string literals, 60–62
dynamically allocating arrays, 31–32
dynamic_cast( ), 360–361, 453

enable_shared_from_this, 191
encodings, UCS, 665
end( ) function, 523
#endif directive, 6, 367–368
enumerated types, 15–16, 264–265
  strongly typed enumerations, 16
  structs, 16–17
eof( ) method, 414–415
equal( ) algorithm, 525, 634–635
equal_range( ) algorithm, 649
erf( ) function, 511
error checking, containers, 539
error handling, 433
  constructors, 462–464
  destructors, 467
  exceptions, 434–435
  input streams, 418–419
  interviews, 1053–1054
  memory allocation errors, 459–461
  output streams, 414–415
  error logging, debugging and, 935–937
exception handling, futures, 846–847
exceptions, 433
  catching, 437–439, 976
    in class hierarchy, 448–449
    const reference, 440–441
    multiple, 441–443
  classes, writing, 449–452
  cleanup and, 458
  copying, 824–826
  custom, 88
  exception specification, 446
  header files, 1081
  nested, 452–454
  noexcept keyword, 445–446
  rethrowing, 454–456, 458, 824–826
  standard, 446
    hierarchy, 446–448
  throw list, 446
  throwing, 437–439
    multiple, 441–443
  throwing exceptions, 976
types, 439–440
  uncaught, 444–445

earlybinding, 287
efficiency
  design-level, 882, 889
  caching, 889–890
  object pools, 890–894
  interviews, 1058–1059
  language, 882–883
  language-level, 882, 883–884
    inline functions, 888–889
    inline methods, 888–889
    memory allocation, 888
  object handling, 884–888
  profiling and, 894
    example, Visual C++ 2017, 902–906
    gprof and, 895–902
  resources, 1071
  elements, value semantics, 537–538
  elliptic integrals, 511
emplace( ) method, 580
emplace operations, containers, 554
employee records program, 46
Database class
  Database.cpp file, 51–52
  Database.h file, 50–51
  DatabaseTest.cpp file, 52–53
Employee class
  Employee.cpp file, 48–49
  Employee.h file, 47–48
  EmployeeTest.cpp file, 50
user interface, 53–55
```
exchange( ) algorithm, 647
exclusive_scan( ) algorithm, 657
exceptions, 37–38, 434–435
    Standard Library, 510
execution policy, parallel algorithms, 655
exp( ) function, 511
explicit keyword, 391
expressions, folding expressions, 796–797
extern keyword, 349–350
extraction operators, overloading, 485–486

F

factory patterns, 997–999, 1002–1003
    implementing, 999–1001
    other uses, 1003–1004
fail( ) method, 414–415
    [[fallthrough]] attribute, 364
false-sharing, multithreaded programming
    and, 818
fgets( ) function, 411
file streams, 426–427
    binary mode, 427–428
fgets( ) function and, 411
    <fstream> header file, 426–427
fprintf( ) function and, 411
fputs( ) function and, 411
fread( ) function and, 411
fscanf( ) function and, 411
fwrite( ) function and, 411
ios_base::app constant, 427
ios_base::ate constant, 427
ios_base::binary constant, 427
ios_base::in constant, 427
ios_base::out constant, 427
ios_base::trunc constant, 427
text mode, 427–428
files
    header files, 5
    reading from, 976–977
    source files, 5
    <fstream> header file, 426–427
filesystem support library, 513
directory entry, 721
directory iteration, 722–723
    helper functions, 721–722
    path, 720–721
fill( ) algorithm, 526
fill_n( ) algorithm, 526
find( ) algorithm, 524, 608–611
find_end( ) algorithm, 524
find_first_if( ) algorithm, 524
find_if( ) algorithm, 524, 608–611
find_if_not( ) algorithm, 524
findMatches( ) function, 355
floor( ) function, 511, 695
flush( ) method, 410, 413
    flush-on-access, streams, 430–431
fma( ) function, 511
for loop, 27
for_each( ) algorithm, 646
for_each_n( ) algorithm, 646
forward declarations, 261, 368–369
    forward_list container, 515, 520
    forwards_list container, 566–568
fprintf( ) function, 411
fputs( ) function, 411
frameworks, object-oriented
    MFC (Microsoft Framework Classes), 988
    MVC (Model-View-Controller), 989–990
fread( ) function, 411
freeing memory, 235–236
friends, 232–233
fscanf( ) function, 411
full class template specialization, 782–785
function call operator, overloading, 491–492
function objects, 513, 607
    arithmetic, 621
    bitwise, 624
    comparison, 622–623
    invokers, 629
    logical, 623–624
    threads and, 820–822
    transparent, 622
    writing, 629–630
function pointers
    threads and, 819–820
type aliases, 353–355
function templates, 400–401
class templates and, 403–404
overloading, 402–403
   specialization and, 403
return types, 405–406
specialization, 401–402
functional composition, adaptor function objects and, 624
functional relationships, 135
functionality, 130–131
functions, 21, 374
   abs( ), 511
   at( ), 376
   atan2( ), 511
   begin( ), 523
cbegin( ), 523
   ceil( ), 511
cend( ), 523
cleanup( ), 248
crbegin( ), 523
createDocArray( ), 169
crend( ), 523
current function name, 23
declarations, 5, 21–22
definitions, 5
   [[deprecated]] attribute, 364
doSOMething( ), 453
doWorkInThread( ), 825
DumpRange( ), 652
dend( ), 523
erf( ), 511
exp( ), 511
fgets( ), 411
findMatches( ), 355
floor( ), 511
fma( ), 511
fprintf( ), 411
fputs( ), 411
fread( ), 411
friends, 232
fscanf( ), 411
fwrite( ), 411
setValue( ), 201
handleMessage( ), 244
handleValue( ), 793
hash functions, 591–593
increment( ), 828–829
intEqual( ), 355
log( ), 511, 939
main( ), 6–7
make_pair( ), 389–390
malloc( ), 167
   [[maybe_unused]] attribute, 365
member functions, calling, 628–629
mem_fn( ), 628–629
new handler callback, 460–461
   [[nодiscard]] attribute, 364–365
   [[noreturn]] attribute, 363–364
parameters, 374
polymorphic function wrapper, 612
predicate function callbacks, 610
pow( ), 511
processValues( ), 792–793
pthread_create( ), 819
rand( ), 512, 702
rbegin( ), 523, 730
realloc( ), 17
rethrow_nested( ), 453
return type, deduction, 22–23
set_terminate( ), 445
setValue( ), 201
sin( ), 511
sinh( ), 511
sprintf( ), 411
sscanf( ), 411
sqrt( ), 511
srand( ), 512, 702
sscanf( ), 411
swap( ), 247–248
terminate( ), 444
tgamma( ), 511
threadFunc( ), 825
function-try-blocks, 464–467
functors, 607. See function objects
<future> folder, 513
futures, 843
   exception handling, 846–847
promises, 843
std::async, 845–846
std::future, 843–844
std::packaged_task, 844–845
std::promise, 843–844
std::shared_future, 847–848
threads, 824
fwrite( ) function, 411

G
GameBoard class, error handling and, 468–472
generate( ) algorithm, 526
generate_n( ) algorithm, 526
generic programming, 374, 378
   templates and, 508
get_money input manipulator, 423
getStatus( ), 85
getters, 139
generate_n( ) algorithm, 526
get_value input manipulator, 423
gValue( ) function, 201
global scope, 36
get( ) method, 414–415
GPUs (graphics cards), 814
granularity of tests, 916
Grid class, 378, 380–381
   definition, 379–380
   methods, 380–381

H
handleMessage( ) function, 244
handles, 146
handleValue( ) function, 793
has-a relationships, 129, 132–135
hash functions, 591–593
hash tables, 518–519, 591–598
header files, 5, 1075–1077
   <algorithm>, 630–631
   algorithms, 1078–1079
allocators, 1078–1079
<any>, 713–714
<array>, 516
<atomic>, 827
<chrono>, 512
<condition_variable>, 840
containers, 1077–1078
<deque>, 516
duplicate definitions, 367–368
exceptions, 1081
<filestream>, 426–427
forward declarations, 368–369
<forward_list>, 515
<functional>, 621
<future>, 513
general utilities, 1079–1080
initializer lists, 28
<initializer_list>, 512
I/O streams, 1081–1082
<iostream>, 5, 412
iterators, 1078–1079
<list>, 515
<locale>, 509
<map>, 518
mathematical utilities, 1081
<multimap>, 518
<multiset>, 517–518
<numerate>, 630
<optional>, 711
<ostream>, 412
<priority_queue>, 516–517
<queue>, 516
<ratio>, 691–694
<set>, 517–518
<stack>, 517
cstdintr>, 514
<string>, 509
template definitions, 384–385
<thread>, 513
threading support library, 1082
<tuple>, 512
<type_traits>, 514
<utility>, 512, 630
heap
  constructors on, 210
  new keyword, 165
  objects, 207
  pointers, 30–31
Hello World, 3
helper functions, filesystem support library,
  721–722
hex input manipulator, 423
hex output manipulator, 415
hierarchies
  catching exceptions, 448–449
  standard exceptions, 446–448
high_resolution_clock class, 698–699
Hungarian Notation, 85
_i literal, 366
#ifdef directive, 6
if/else statements, 17–18
#endif directive, 6, 367–368
Impl class, 273
implementation class, 272–275
implicit conversions, 267
#include directive, 6, 368
inclusive_scan( ) algorithm, 657
increment( ) function, 828–829
inheritance
  base classes, 130
    virtual, 331–332
  casts, 360–361
  constructors, 316–319
  derived classes
    assignment operator, 327–329
    copy constructors, 327–329
  extending classes, 278–279
    Derived class, 279–281
functionality, 130–131
multiple, 137–138, 308–309
  base classes, ambiguous, 311–312
  naming collisions, 309–310
uses, 312
non-public, 331
overridden methods
  base class method, 324–327
  base class method overloaded, 321–322
  base class method private, 322–324
  base class method protected, 322–324
  base class method static, 320–321
  method return type, 313–315
  parameters, changing, 315–316
parent classes, 130
polymorphism and, 301–308
preventing, 281
properties, 131
reuse and, 291–300
RTTI (run-time type information),
  329–330
specialization comparison, 399
superclasses, 130
templates, 397–398
inheritance techniques, interviews,
  1050–1051
initialization
  constructors, 215–218
RAII (Resource Acquisition is Initialization),
  979–981
uniform, 43–45
copy list, 45
direct list, 45
  variables, non-local, 351
initializer lists, 28
initializer-list constructors, 220–222
<initializer_list> header, 512
inline keyword, 256
inline methods, 255–257
efficiency and, 888–889
inline variables, static data member access,
  259
in-memory stream, 425
input streams
  >> operator, 417–418
error handling, 418–419
manipulators, 423
  methods
<table>
<thead>
<tr>
<th>Insert – ios_base::in constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>bad( ), 418</td>
</tr>
<tr>
<td>fail( ), 418</td>
</tr>
<tr>
<td>get( ), 419</td>
</tr>
<tr>
<td>getline( ), 422</td>
</tr>
<tr>
<td>good( ), 418</td>
</tr>
<tr>
<td>peek( ), 421–422</td>
</tr>
<tr>
<td>putback( ), 421</td>
</tr>
<tr>
<td>unget( ), 420–421</td>
</tr>
<tr>
<td>objects and, 423–425</td>
</tr>
<tr>
<td>insert() method, 578–579, 732</td>
</tr>
<tr>
<td>insertion operators, overloading, 485–486</td>
</tr>
<tr>
<td>insert_or_assign() method, 579–580</td>
</tr>
<tr>
<td>instances, 125, 201</td>
</tr>
<tr>
<td>int* type declaration, 352–353</td>
</tr>
<tr>
<td>integration, 929</td>
</tr>
<tr>
<td>JSON-based file serializer, 928</td>
</tr>
<tr>
<td>shared resources, readers/writers, 928</td>
</tr>
<tr>
<td>wrapper, third-party library, 928–929</td>
</tr>
<tr>
<td>intEqual() function, 355</td>
</tr>
<tr>
<td>interface class, 272–275</td>
</tr>
<tr>
<td>interfaces</td>
</tr>
<tr>
<td>abstraction and, 138–139</td>
</tr>
<tr>
<td>APIs (application programming interface), 140</td>
</tr>
<tr>
<td>audience and, 139</td>
</tr>
<tr>
<td>component interface, 140</td>
</tr>
<tr>
<td>containers, 514</td>
</tr>
<tr>
<td>libraries, 140</td>
</tr>
<tr>
<td>planning, 141</td>
</tr>
<tr>
<td>purpose, 139</td>
</tr>
<tr>
<td>reusable, 153</td>
</tr>
<tr>
<td>comments, 156</td>
</tr>
<tr>
<td>customizability, 157</td>
</tr>
<tr>
<td>documentation, 156</td>
</tr>
<tr>
<td>ease of use and generality, 157–158</td>
</tr>
<tr>
<td>familiarity, 153–154</td>
</tr>
<tr>
<td>general-purpose, 157</td>
</tr>
<tr>
<td>ISP (Interface Segregation Principle), 158</td>
</tr>
<tr>
<td>multiple, 158</td>
</tr>
<tr>
<td>operator overloading, 154</td>
</tr>
<tr>
<td>required functionality, 155</td>
</tr>
<tr>
<td>uncluttered, 155–156</td>
</tr>
<tr>
<td>subsystems, 140</td>
</tr>
<tr>
<td>utility classes, 140</td>
</tr>
<tr>
<td>interviews</td>
</tr>
<tr>
<td>applications, 1061</td>
</tr>
<tr>
<td>classes, 1046–1050</td>
</tr>
<tr>
<td>coding style, 1041–1042</td>
</tr>
<tr>
<td>debugging, 1060</td>
</tr>
<tr>
<td>design</td>
</tr>
<tr>
<td>frameworks and, 1060</td>
</tr>
<tr>
<td>object-oriented design, 1044–1045</td>
</tr>
<tr>
<td>patterns, 1060–1061</td>
</tr>
<tr>
<td>program design, 1042–1044</td>
</tr>
<tr>
<td>reuse and, 1045</td>
</tr>
<tr>
<td>efficiency, 1058–1059</td>
</tr>
<tr>
<td>error handling, 1053–1054</td>
</tr>
<tr>
<td>inheritance techniques, 1050–1051</td>
</tr>
<tr>
<td>I/O, 1053</td>
</tr>
<tr>
<td>library utilities, 1056</td>
</tr>
<tr>
<td>multithreaded programming, 1057</td>
</tr>
<tr>
<td>objects, 1046–1050</td>
</tr>
<tr>
<td>obscure questions, 1051–1052</td>
</tr>
<tr>
<td>operator overload, 1054–1055</td>
</tr>
<tr>
<td>regular expressions, 1040–1041</td>
</tr>
<tr>
<td>software engineering methods, 1057–1058</td>
</tr>
<tr>
<td>Standard Library, 1039–1040, 1055–1056</td>
</tr>
<tr>
<td>strings, 1040–1041</td>
</tr>
<tr>
<td>localization, 1040–1041</td>
</tr>
<tr>
<td>views, 1040–1041</td>
</tr>
<tr>
<td>templates, 1052–1053</td>
</tr>
<tr>
<td>testing abilities, 1059</td>
</tr>
<tr>
<td>IntPtr, 352–353</td>
</tr>
<tr>
<td>ints array, 340–341</td>
</tr>
<tr>
<td>invoke( ) adaptor, 629</td>
</tr>
<tr>
<td>I/O</td>
</tr>
<tr>
<td>bidirectional, 431–432</td>
</tr>
<tr>
<td>interviews, 1053</td>
</tr>
<tr>
<td>I/O streams, 7</td>
</tr>
<tr>
<td>header files, 1081–1082</td>
</tr>
<tr>
<td>Standard Library, 510</td>
</tr>
<tr>
<td>I/O streams and strings, resources, 1065–1066</td>
</tr>
<tr>
<td>ios_base::app constant, 427</td>
</tr>
<tr>
<td>ios_base::ate constant, file streams, 427</td>
</tr>
<tr>
<td>ios_base::binary constant, file streams, 427</td>
</tr>
<tr>
<td>ios_base::in constant, file streams, 427</td>
</tr>
</tbody>
</table>
ios_base::out constant, file streams, 427
ios_base::trunc constant, file streams, 427
<iostream> header file, 5, 412
is-a relationships, 130–135
ISP (Interface Segregation Principle), 158, 159
istream_iterator class, 730
iteration, directories, 722–723
iterative processes, 863
iterator adaptors
insert iterators, 731–733
move iterators, 733–734
reverse iterators, 730–731
iterator design pattern, 992–993
iterators, 521–522, 608, 737
BidirectionalIterator, 631
constant iterators, 631
containers, 539–541
ForwardIterator, 631
header files, 1078–1079
InputIterator, 631
mutable, 631
OutputIterator, 631
RandomAccessIterator, 631
streams
istream_iterator class, 730
ostream_iterator class, 729

K

keywords
auto, 40, 613
cast, 35
methods, 345
parameters, 343
pointers, 343–345
references, 345
variables, 343
constexpr, 346
decltype, 40
default, 214–215
explicit, 391
extern, 349–350
inline, 256
new, 165

noexcept, 445–446
override, 284–285
static
data members, 347
linkage, 347–349
methods, 347
throw, 454–456
try, 464–467
virtual, 281–282, 287–290

L

lambda expressions
capture expressions, 618
generic expressions, 617–618
as parameters, 619
as return types, 618–619
Standard Library algorithms
count_if( ), 619–620
generate( ), 620
syntax, 614–617
threads and, 822
language, efficiency and, 882–883
language-level efficiency, 882, 883–884
inline functions, 888–889
inline methods, 888–889
memory allocation, 888
object handling
catch exceptions, 886
move semantics, 886
pass-by-reference, 884–885
return-by-reference, 886
return-value optimization, 887–888
temporary objects, 886–887
late binding, 287
Legender polynomials, 511
lexicographical_compare( ) algorithm, 525,
634–635
libraries
filesystem support
directory entry, 721
directory iteration, 722–723
helper functions, 721–722
path, 720–721
interfaces, 140
open-source
  BSD (Berkeley Software Distribution), 113
  CC (Creative Commons) license, 113
  CPOL (Code Project Open License), 113
  GPL (GNU Public License), 113
  locating, 113
    use guidelines, 113–114
  Ratio library, 691–694
library utilities, interviews, 1056
linked lists, 515
linking streams, 430–431
list container, 515, 520, 562–566
lists
  doubly linked, 515
  singly linked, 515
literal operators, 366–367
literal types, 346
literals, 10
  _i literal, 366
  _s literal, 366
string literals, localization and, 664
user-defined, 365–366
  standard, 367
little-endian ordering of bytes, 1021
local storage, threads, 823
<locale> header, 509, 667–668
localization, 509, 663–664
  ASCII characters and, 663–664
  character classification, 669–670
  character conversion and, 670
  character sets, 664
    matches, 674–676
    non-Western, 665
  conversions, 667–668
  facet categories, 670–671
  facets, 668
  locales, 668–669
string literals, 664
  wide characters, 664–665
log( ) function, 511, 939
Logger class, 938
logical evaluation operators
  !, 21
  !=, 20
  &c&, 21
  <=, 20
  ==, 20
  >, 20
  >=, 20
logical expressions, short-circuit logic, 21
logical function objects, 623–624
loops
  for, 27
  do/while, 27
  while, 26–27
lower_bound( ) algorithm, 649
LSP (Liskov Substitution Principle), 134, 158
lvalues, 243–245
macros, preprocessor and, 371–372
main( ) function, 6–7
make_pair( ) function, 389–390
malloc( ) function, 167
man pages, 156
map container, 518
  element look up, 581–582
  element removal, 582
  example, 583–585
  inserting elements, 578–580
  iterators, 580–581
  nodes, 582–583
map multimap container, 521
MatchFunction, 355
mathematical utilities, Standard Library, 511
  Bessel functions, 511
  beta functions, 511
  complex numbers, 511
  cylindricalNeumann functions, 511
  elliptic integrals, 511
  Legender polynomials, 511
  numeric limits and, 511
  ratio class, 511
  ratio class template, 511
  valarray class, 511
max( ) algorithm, 653
[[maybe_unused]] attribute, 365
member functions
calling, 628–629
threads and, 823
mem_fn( ) function, 628–629
memory
allocating
failure, 167–168
new keyword, 166–167
deallocating, delete keyword, 166–167
decomstructors, 235–236
double deleting, 197
dynamic, 164–165
freeing, 235–236
garbage collection, 181–182
heap, 29
new keyword, 165
leaks, 194–197, 238
low-level operations, 179–182
malloc( ) function, 167
object pools, 182
orphaned, 238
out-of-bounds, accessing, 193
pointer arithmetic, 179–180
pointers, 30–31
dangling, 197
polymorphic memory wrappers, 728–729
stack, 29
stack frame, 29
strings, underallocating, 192–193
memory allocation
errors, custom failure behavior, 460–461
operators, overloading
delete expression, 501
delete operator, 500–506
new expression, 501
new operator, 500–506
pre-allocating, 888
memory dumps, 946–947
memory management, 163–164
custom, 180–181
interviews, 1045–1046
memory pool, 728
merge( ) algorithm, 652
meta-information, coding style and, 75–76
metaprogramming templates, 797–798
loop unrolling, 799–800
tuples, printing, 800–801
compile-time integer sequence, 803
constexpr if statement, 802–803
type traits, 803–805
constexpr if statement, 810–811
enable_if, 808–810
logical operators, 811
type categories, 805–807
type relations, 807–808
methodPtr variable, 355–356
methods, 412–413
:: (scope resolution operator), 204
access, 204
allocate( ), 728–729
assign( ), 545
bad( ), 414–415, 418
base( ), 731
calling others, 204–206
clone( ), 375
const, 251–252
copy, 218–219
default, 212–215
delegating, 211, 222
on the heap, 210
initializer-list constructors, 220–222
initializers, 215–218
multiple, 211
overloading, 211
on the stack, 210
writing, 209–210
copy( ), 732
deallocate( ), 728
decomstructors, 224–225
emplace( ), 580
eof( ), 414–415
fail( ), 414–415, 418
flush( ), 410, 413
min – mutable data members

good( ), 414–415, 418
hiding, 286–290
inline, 255–257
insert( ), 578–579, 732
insert_or_assign( ), 579–580
method templates, 391–393
    non-type parameters, 393–395
moveFrom( ), 247
mutable data members, 253
operator[ ], 580
overloading, 253–254
    const and, 254–255
    deleting explicitly, 255
overriding, 281–282
    override keyword, 284–285
    preventing, 290
    syntax, 282–283
pointers, type aliases, 355–356
pure virtual methods, 303
put( ), 413
resize( ), 777
seek( ), 428–430
static, 251
std::as_const( ), 358
str( ), 425
swap( ), 546
tell( ), 428–430
this pointer, 206–207
tie( ), 430–431
write( ), 413
min( ) algorithm, 653
minimum-maximum algorithms, 653–654
mismatch( ) algorithm, 525, 634–635
mixin classes, 138, 985, 988
    designing, 986–987
    implementing, 987
modifying sequence algorithms, 525–526,
    636–644
copy( ), 638–639
move( ), 640–641
remove( ), 641–643
replace( ), 641
reverse( ), 644
sample( ), 643–644
shuffle( ), 644
transform( ), 637–638
unique( ), 643
move( ) algorithm, 526
move assignment operators, 243, 245–248,
    481
move constructor, 245–248
move semantics, 190, 243, 612, 733
    implementing, 245–246
    object handling and, 886
move_backward( ) algorithm, 526
moveFrom( ) method, 247
multicore processors, 814
multi-dimensional arrays
    heap, 173–175
    stack, 172–173
multimap container, 518
multiple inheritance, 137–138, 308–309
multiset container, 517–518
multithreaded programming, 814
    condition variables, 840–842
    data races, 815–817
deadlocks, 817–818
debugging, 956–957
false-sharing, 818
=future> header, 513
futures, 843
    exception handling, 846–847
    promises, 843
    std::async, 845–846
    std::future, 843–844
    std::packaged_task, 844–845
    std::promise, 843–844
    std::shared_future, 847–848
interviews, 1057
logger class example, 848–853
mutual exclusion, mutex classes, 832–840
race conditions, 815–817
resources, 1073
tearing, 817
=future> header, 513
thread pools, 853
threading design, 853–855
mutable data members, 253
mutable iterators, 631
mutex classes, 831
double-checked locking, 839–840
locks
  lock_keyguard, 833–834
  multiple, 835
  scoped_lock, 835–836
  shared_lock, 835
  unique_lock, 834
non-timed
  exclusive ownership, 832
  lock( ) method, 832
  read locks, 832
  readers-writers lock, 832
  shared lock ownership, 832
  shared ownership, 832
  shared_mutex class, 832
  try_lock( ) method, 832
  unlock( ) method, 832
std::call_once, 836–837
thread writing, safe streams, 837–838
timed, 832–833
timed locks, 838–839
mutual exclusion, mutex classes, 831
  non-timed, 832
  timed, 832–833

N
namespaces, 8–10
  aliases, 10
  anonymous, 349
  nested, 9–10
naming
  conventions
    capitalization, 86
    counters, 84
    get_status( ), 86
    getters, 86
    Hungarian Notation, 85
    namespaced constants, 86
    prefixes, 84–85
    set_status( ), 86
    setters, 86
  selecting, 83–84
  negative tests, 923
  nested classes, 263–264
  nested exceptions, 452–454
  nested namespaces, 9–10
  new[ ], 169
  new handler callback function, 460–461
  new keyword, 165
    allocating memory, 166–167
  new operator, 500–506
  noboolalpha input manipulator, 423
  noboolalpha output manipulator, 415
  node handles, 582–583
  node-based data structures, 582–583
  [[nodiscard]] attribute, 364–365
  noexcept keyword, 445–446
  none_of( ) algorithm, 525, 636
  non-local variables
    destruction, 351
    initialization order, 351
  non-modifying algorithms, 523–524
  non-modifying sequence algorithms
    comparison algorithms, 634–635
    counting algorithms, 636
    search algorithms, 631–633
    specialized searchers, 633
  non-timed mutex classes
    exclusive ownership, 832
    lock( ) method, 832
    read locks, 832
    readers-writers lock, 832
    shared lock ownership, 832
    shared ownership, 832
    shared_mutex class, 832
    try_lock( ) method, 832
    unlock( ) method, 832
  non-type parameters, templates,
    387–388
    method templates, 393–395
  [[noreturn]] attribute, 363
  noshowpoint output manipulator, 415
  noskipws input manipulator, 423
  not1( ) negator adaptor, 627
  not2( ) negator adaptor, 627
not_fn( ) negator adaptor, 626–627
NRVO (named return value optimization), 888
null pointers, 30–33
<numERIC> header, 630

O

object files, linking, 5
object handling
  catch exceptions, 886
  move semantics, 886
  return-by-reference, 886
  return-value optimization, 887–888
  temporary objects, 886–887
objects, 127, 374
  assignment operators
    declaring, 225–226
    defining, 226–227
    explicitly defaulted, 227
    explicitly deleted, 227
  block ends, 224
  casting, 299–300
  constructors, 209
    compiler-generated, 222–223
    copy, 218–219
    default, 212–215
    delegating, 211, 222
    on the heap, 210
    initializer-list constructors, 220–222
    initializers, 215–218
    multiple, 211
    overloading, 211
    on the stack, 210
    writing, 209–210
  creating, 208–209
  data members, moving, 247
  destructors, 224–225
  functors, 513, 607
    on the heap, 207–208
  input, 423–425
  interviews, 1046–1050
  out of scope, 224
  output, 423–425
  over-objectification, 127–128
  passing by reference, 220
    as return values, 228–229
    on the stack, 207
  obscure questions in interviews, 1051–1052
  observer pattern, 1014–1016
  OCP (Open/Closed Principle), 158
  oct input manipulator, 423
  oct output manipulator, 415
  one-dimensional arrays, 24
  OOP (object-oriented programming), 40, 124
    aggregation, 129
    behaviors, 126
    classes, 124–125
      base classes, 130
      instances, 125
    collections, 126
    components, 125
deriving, 130
  extending, 130
  frameworks, 988–989
  general, 128
  hierarchies, 136
  inheritance, 130
    base classes, 130
    functionality, 130–131
    multiple, 137–138
    parent classes, 130
    properties, 131
    superclasses, 130
    instances, 125
  MVC (Model-View-Controller), 990
  objects, 127, 374
    general, 128
    over-objectification, 127–128
  polymorphism, 131–132
  properties, 125
  relationships
    functional, 135
    has-a, 129, 132–135
    is-a, 132–135
    is-a relationship, 130–132
    mixin classes, 138
    shares-with, 138
opaque classes – operators

SOLID principles, 158–159
state, 127–128
subclasing, 130
opaque classes, 146

open-source libraries
BSD (Berkeley Software Distribution), 113
CC (Creative Commons) license, 113
CPOLe (Code Project Open License), 113
GPL (GNU Public License), 113
locating, 113
use guidelines, 113–114
operating systems, resources, 1073
operational algorithms, 646
operator[ ] method, 580
operator overload
add( ) method, 265
argument types, 476
arithmetic operators, 269–270
decrement, 483–484
increment, 483–484
unary minus, 483
unary plus, 483
behavior, 477
binary logical operators, 484–485
bitwise operators, 484–485
comparison, 270–271
conversions, implicit, 267
dereferencing operators, 492–494
operator->*, 495–496
operator * , 494
operator ->, 494
extraction operators, 485–486
function call operator, 491–492
global functions, 475–476
insertion operators, 485–486
interviews, 1054–1055
limitations, 474–475
memory allocation operators, 500–506
delete expression, 501
delete operator, 500–506
new expression, 501
new operator, 500–506
methods, 475–476
move assignment operators, 481
operator+, 266–269
operators not to overload, 477–478
overloadable operators, 478–481
reasons, 474
relational operators, 482
return types, 477
rvalue references, 481–482
Standard Library, 509
subscripting operator, 486–489
non-integral array indices, 490
operator[ ], 489–490
type building and, 271–272
operators
arity, 475
assignment operators, 240–242
copy, 228
declaring, 225–226
defaulted, explicit, 227
defining, 226–227
deleted, explicit, 227
move, 243
associativity, 475
binary, 13
conditional statements, 20
conversion, 496–497
   Boolean expressions, 498–500
   explicit, 497
delete, 32, 500–506
literal, 366–367
logical evaluation
   !, 21
   !=, 20
   &&, 21
   <, 20
   <=, 20
   ==, 20
   >, 20
   >=, 20
   |, 21
move assignment operators, 481
new, 31–32, 500–506
precedence, 475
relational, 482
ternary, 13
unary, 13
<optional> header, 711
ordered associative containers
map
  constructing, 577–578
element look up, 581–582
element removal, 582
element look up example, 583–585
inserting elements, 578–580
iterators, 580–581
nodes, 582–583
multimap, 585–588
multiset, 590
set, 589–590
orphaned memory, 238
orthogonality, 521
<ostream> header file, 412
ostream_iterator class, 729
output streams, 411–412
cout, 412
error handling, 414–415
manipulators, 415–417
methods, 412–413
  bad(), 414–415
eof(), 414–415
fail(), 414–415
flush(), 413
good(), 414–415
put(), 413
write(), 413
objects and, 423–425
raw output, 413
overload resolution, 254
overloading, 211
  methods, 253–254
    const and, 254–255
    deleting explicitly, 255
overriding keyword, 284–285
overriding methods, 281–282
overriding keyword, 284–285
preventing, 290
syntax, 282–283
pair class, 576–577
parallel algorithms, 655
parameter packs, 793
parameterization, 374
deduction, 404–405
types, 374
values, 374
parameters, 257, 374
const keyword, 343
const keyword and, 35
cooked-mode literal operators, 366
references, 336–337
  pass-by-reference, 337–338
  pass-by-value, 337–338
from pointers, 337
template template parameters, 778–780
templates, 386
deduction for constructors, 389–391
default values, 389
non-type parameters, 387–388, 780–782
type parameters, 778–780
parent classes, 130, 278
partial class template specialization, 782–787
partial specialization, templates, 397
partition algorithms, 647–648
partition_copy() algorithm, 648
passing, by reference, 220
pimpl idiom, 272
platform, 1017
  binary compatibility, 1019–1020
pointer arithmetic, 179–180
pointers, 30–31
  aliasing, 183
  arrays as, 177–178
  casting with, 176
  const keyword, 343–345
dangling, 197, 238
to data members, type aliases, 355–356
dereferencing, 30, 175–176
function pointers, type aliases, 353–355
to methods, type aliases, 355–356
models, 175–176
null, 30
  constant, 32–33
reference counting, 183
to references, 336
*versus* references, 87, 339–343
shared ownership, 183
smart, 33–34
  auto_ptr, 192
  enable_shared_from_this, 191
move semantics, 190
referencing counting and, 188–189
shared_ptr, 186–188
unique_ptr, 183–186
weak_ptr, 189–190
smart pointers, 182–183
on the stack, 210
to structures, 31
this, 206–207
polymorphic function wrapper, 612
polymorphic memory wrappers, 728–729
polymorphism, 131–132
  inheritance and, 301–308
pow( ) function, 511
#pragma directive, 6
#pragma once directive, 368
predicate function callbacks, 610
preprocessor, 5
  directives, 6
  macros, 371–372
priority_queue container, 516–517, 520
  function objects and, 622–623
private implementation idiom, 272
private nested classes, 264
procedural approach, 124
procedures, 374
process, software engineering and, 860–861
processors, multicore, 814
processValues( ) function, 792–793
profiling, 894
gprof and, 895–902
programming, 374. *See also* design; OOP
  (object-oriented programming)
functions, 374
generic, 378
multithreaded, 814
objects, 374
parameters, 374
procedures, 374
style, resources, 1070–1071
templates, 374
properties, 125, 131
protected nested classes, 264
proxy patterns, 1004–1006
pseudo-random number engines
  linear congruential, 704
  Mersenne twister, 704
  subtract with carry engine, 704
pthread_create( ) function, 819
ptrRef, 336
pure virtual methods, 303
push_back( ) call, 734
put( ) method, 413
put_money output manipulator, 415
put_time output manipulator, 423
quoted input manipulator, 423
quoted output manipulator, 415
QA (quality assurance), 909
queue container, 516, 520
race conditions, 815–817
RAII (Resource Acquisition Is Initialization), 979–981
rand( ) function, 512, 702
random numbers
distributions
  Bernoulli, 708
  normal, 708–709
  Poisson, 708
random_shuffle – regular expressions

sampling, 709
uniform, 708
engines, 702–704, 707–711
adapter pattern, 705
adaptors, 705–706
base engine, 705
linear_congruential_engine, 703
mersenne_twister_engine, 703
predefined, 705–706
random_device, 703
subtract_with_carry_engine, 703
generate( ) algorithm, 707
generating, 702–703, 706–707
pseudo-random number engines
linear congruential, 704
Mersenne twister, 704
subtract with carry engine, 704
resources, 1068
Standard Library, 512
random_shuffle( ) algorithm, 526
range-based for loop, 27
<ratio> header, 691–694
Ratio library, 691–694
raw mode literal operators, 366
raw pointers, 33–34
raw string literals, 60–62
raw-mode literal operators, 367
rbegin( ) function, 523, 730
reading from files, 976–977
realloc( ) function, 170
recursive_directory_iterator, 722
ref( ) function, 537
refcall( ) function, 339
reference data members, 261–262
const, 262–263
reference variables, 334–335
references, 35–36, 334
const keyword, 345, 440–441
data members, 336
lvalues, 243–245
modifying, 335
parameters, 336–337
pass-by-reference, 337–338
pass-by-value, 337–338
from pointers, 337
passing by, 36–37, 220
passing by const reference, 37
to pointers, 336
return values, 338
rvalues, 243–245, 338–339
versus pointers, 87, 339–343
reference_wrapper class, 537
regression testing, 930
regular expressions
alternation, 673
anchors, 673
back references, 673, 677
capture groups, 673
character sets, 674–676
character classes, 675–676
range specification, 675
decision making and, 671
ECMAScript, 672
grammars, 672
grouping, 673
interviews, 1040–1041
iteration and, 671
lookahead, 677
matches, 672
parsing and, 671
patterns, 672
precedence, 674
raw string labels, 677–678
regex library, 678–679
regex_iterator( ) algorithm, 683–684
regex_match( ) algorithm, 679–682
regex_replace( ) algorithm, 687–690
regex_search( ) algorithm, 682–683
regex_token_iterator, 685–687
repetition, 673–674
replace, 672
replace operations, 673
searches, 672
Standard Library, 510
subexpressions, 673
tokenization and, 671
transformation and, 671
validation and, 671
reinterpret_cast – reusable code

wildcards, 673
word boundaries, 676–677
reinterpret_cast( ), 359–361
relational operators, 482
relationships
  functional, 135
  has-a, 129, 132–135
  is-a, 130–135
  mixin classes, 138
  shares-with, 138
remainder( ) function, 511
remove( ) algorithm, 526
remove_copy( ) algorithm, 526
remove_copy_if( ) algorithm, 526
remove-erase-idiom, 642
remove_if( ) algorithm, 526
rend( ) function, 523, 730
replace( ) algorithm, 526
replace_copy( ) algorithm, 526
replace_copy_if( ) algorithm, 526
replace_if( ) algorithm, 526
resize( ) method, 777
resources
  algorithms, 1068
  beginning C++, 1063–1064
  C++11, 1067
  C++14, 1067
  C++17, 1067
  computer architecture, 1071
data structures, 1068
debugging, 1072
design patterns, 1072
efficiency, 1071
general C++, 1064–1065
I/O streams and strings, 1065–1066
multithreaded programming, 1073
operating systems, 1073
programming style, 1070–1071
random numbers, 1068
software
  engineering methodology, 1069–1070
  open-source, 1068–1069
  Standard Library, 1066
templates, 1067
testing, 1071
UML (Unified Modeling Language), 1067
rethrowing exceptions, 454–456, 458
rethrow_nested( ) function, 453
return type functions, 22–23
  function templates, 405–406
return values
  optimization, object handling and, 887–888
  references, 338
return-by-reference, object handling and, 886
reusable code, 101–104
  abstraction and, 145–146
  advantages, 105
  APIs, 104
  big-O notation, 108–109
  capabilities, 107–108
  dependency injection, 103
design
  aggregation, 148
  class hierarchies, 147–148
  high cohesion, 146–148
  interface dependencies, 148
  low coupling, 147
  subsystems, 147
design reuse, 103
design-by-contract, 150
disadvantages, 105–106
derror checking, 150
extensibility and, 151–153
  frameworks, 104
inheritance and, WeatherPrediction class, 291–294
interfaces, 153
  comments, 156
  customizability, 157
documentation, 156
ease of use and generality, 157–158
familiarity, 153–154
general-purpose, 157
  ISP (Interface Segregation Principle), 158
  multiple, 158
  operator overloading, 154
  required functionality, 155
  uncluttered, 155–156
interviews, 1045
libraries, 104
licensing and, 110–111
mottos, 144
open-source libraries
  BSD (Berkeley Software Distribution), 113
  CC (Creative Commons) license, 113
  CPOL (Code Project Open License), 113
  GPL (GNU Public License), 113
locating, 113
open-source software, 112
use guidelines, 113–114
optimal reuse design, 146–153
performance and, 109–110
platforms and, 110
versus polymorphism, 131–132
postconditions, 150
preconditions, 150
prototypes, 111–112
resources, 111
safe code, 150–151
SRP (Single Responsibility Principle), 146–147
stand-alone classes, 104
stand-alone functions, 104
Standard Library, 114
support and, 110–111
templates, 148
  benefits, 149
  versus inheritance, 150
problems, 149
std::vector class, 148
third-party apps, 112
types of code, 104
writing reusable code, 102–103
reverse( ) algorithm, 526
reverse_copy( ) algorithm, 526
ring buffers, debugging and, 942–945
round( ) function, 695
rule of zero, 250
runtime_error, 441–443
rvalues, 243–245, 733
  operator overload and, 481
  references, 338–339
RVO (Return Value Optimization), 229, 888

S

_s literal, 366
sample( ) algorithm, 526
Sashimi Model life cycle model, 863
scan algorithms, 657
scope, 362
global scope, 362
resolution, 362
search( ) algorithm, 524
search algorithms, 524, 631–633
  binary search, 649–650
search_n( ) algorithm, 524
seek( ) method, 428–430
sequence algorithms
  modifying, 636–644
  non-modifying
    comparison algorithms, 634–635
    counting algorithms, 636
    search algorithms, 631–633
    specialized searchers, 633
sequential containers, 773
array, 568–569
deque, 562
forwards_list, 566–568
list, 562–566
vector
  algorithmic complexity, 554
  assigning, 545–546
  comparing, 546
  constructors, 544–545
  copying, 545–546
  destructors, 544–545
  dynamic-length, 544
  elements, 550–553
  fixed-length, 542–543
  iterators, 547–550, 554
  memory allocation scheme, 554–556
  move semantics, 553–554
  reference storage, 550
  round-robin schedule example, 556–561
  vectorbool specialization, 561
set algorithms, 650–653
set container, 517–518
set multiset container, 521
set_difference( ) algorithm, 650
setfill output manipulator, 415
set_intersection( ) algorithm, 650
set_new_handler(), 461
setprecision output manipulator, 415
setStatus(), 85
set_symmetric_difference( ) algorithm, 650
set_terminate( ) function, 445
setters, 139
set_union( ) algorithm, 650
setValue() function, 201
setw output manipulator, 415
shallow copying, 236
shared_ptr pointer, 186–188
shares-with relationships, 138
short circuit logic, 21
showpoint output manipulator, 415
shuffle( ) algorithm, 526
sin( ) function, 511
singleton design pattern, 993–997
singly linked lists, 415
sinh( ) function, 511
skipws input manipulator, 423
smart pointers, 33–34, 182–183
auto_ptr, 192
enable_shared_from_this, 191
move semantics, 190
reference counting and, 188–189
shared_ptr pointer, 186–188
stack unwinding, 457–458
Standard Library, 510
unique_ptr pointer, 182–186
weak_ptr, 189–190
smoke testing, 930
software
design, 96
open-source, resources, 1068–1069
software engineering methods
custom, 877–878
interviews, 1057–1058
process and, 860–861
resources, 1069–1070
RUP (Rational Unified Process), 868–869
Scrum, 869
benefits, 871
Daily Scrum, 870
drawbacks, 872
effort estimation, 870
PO (Product Owner), 870
product backlog, 870
SM (Scrum Master), 870
sprint cycle, 870
Sprint Planning, 870
triage, 876–877
UP (Unified Process)
construction phase, 867
disciplines, 867
elaboration phase, 867
inception phase, 867
iterations, 867
transition phase, 867
XP (Extreme Programming), 872–876
integration, 874
iteration planning, 873
metaphor, 874
pair programming, 872–873
refactoring, 874
release planning, 873
small releases, 874
testing, 873
software life cycle
Agile Model, 866–867
Sashimi Model, 863
Spiral Model, 863
spiral-like models, 863–866
Waterfall Model, 861–863
SOLID principles
DIP (Dependency Inversion Principle), 159
ISP (Interface Segregation Principle), 159
LSP (Liskov Substitution Principle), 158
OCP (Open/Closed Principle), 158
SRP (Single Responsibility Principle), 158
sorting algorithms, 649
Source Code Control software, 878–880
source code servers, 946–947
source files, 5
template definitions, 385–386
specialization versus inheritance, 399
specialized versus specialized searchers, 633
Spiral Model life cycle model, 863
spiral-like models, life cycle, 863–866
spreadsheet application, 200
sqrt( ) function, 511
srand( ) function, 512, 702
SRP (Single Responsibility Principle), 146–147, 158
stack, 29
  constructors on, 210
  objects, 207
stack container, 517, 521
stack frame, 29
stack unwinding, 456–457
smart pointers, 457–458
stakeholders, design and, 96
Standard Library, 46, 114
  algorithms, 507, 521
    accumulate, 611–612
    binary search, 529
    comparison, 524–525
    counting, 525
    find( ), 608–611
    find_all( ), 735–736
    find_if( ), 608–611
  heap, 529–530
  iterators, 521, 737
  lambda expressions, 619–620
  minimum-maximum, 530
  modifying sequence, 525–526
  move semantics, 612
  non-modifying, 523–524
  numerical processing, 530–531
  operational, 527
  partition, 527–528
  permutation, 532
  search, 524
  set, 529
  sorting, 528
  swap and exchange, 527
  writing, 735–737
any class, 513
C, 5
containers, 507, 514
  allocators, 760
  array, 516, 520
  associative, 536, 760–773
  bitset, 519, 521, 600–604
  container adapters, 536
  deque, 516, 520
delement requirements, 537–538
error checking, 539
forward_list, 515, 520
hash maps, 738–760
hash tables, 536, 591–598
  iterators, 539–541
  list, 515, 520
  map, 518
  map multimap, 521
  multimap, 518
  multiset, 517–518
  ordered associative containers, 576–590
  priority_queue, 516–517, 520
  queue, 516, 520
  reversible, 760
  sequential, 536, 542–569, 737
  set, 517–518
  set multiset, 521
  stack, 517, 521
  standard C-style arrays, 598–599
  strings, 599–600
  unordered associative containers, 536, 591–598
  unordered_map, 521
  unordered_multimap, 521
  unordered_multiset, 521
  unordered_set, 521
vector, 514–515, 519
  writing, 737–773
exceptions, 510
extending, 735
extensibility, 532
filesystem support library, 513
  function objects, 513
  hash tables, 518–519
header files, 1075–1077
  algorithms, 1078–1079
  allocators, 1078–1079
containers, 1077–1078
exceptions, 1081
general utilities, 1079–1080
I/O streams, 1081–1082
iterators, 1078–1079
mathematical utilities, 1081
threading support library, 1082
heterogeneous elements, 512
homogeneous elements, 512
initializer lists, 512
integer types, standard, 514
interviews, 1039–1040, 1055–1056
I/O streams, 510
mathematical utilities, 511
abs( ) function, 511
atan2( ) function, 511
Bessel functions, 511
beta functions, 511
ceil( ) function, 511
cylindricalNeumann functions, 511
efficient integrals, 511
erf( ) function, 511
exp( ) function, 511
floor( ) function, 511
fma( ) function, 511
Legender polynomials, 511
log( ) function, 511
numeric limits and, 511
pow( ) function, 511
ratio class, 511
remainder( ) function, 511
sin( ) function, 511
sinh( ) function, 511
sqrt( ) function, 511
tgamma( ) function, 511
valarray class, 511
missing functionality, 532
modifying sequence algorithms, 525–526
multithreading, 513
operator overloading, 509
optional class, 512–513
orthogonality, 521
pair template, 512
random number library
rand( ) function, 512
srand( ) function, 512
regular expressions, 510
resources, 1066
smart pointers, 510
stream iterators
istream_iterator class, 730
ostream_iterator class, 729
streams, 510
strings, 509
templates, 508–509
time utilities, <chrono> header file, 512
<tuple> header file, 512
type aliases, 353
type traits, 513
unordered associative containers, 518–519
<utility> header, 512
variant class, 513
standard user-defined literals, 367
state, 128
statements
conditional
if/else, 17–18
operator, 20
switch, 18–19
switch, [[fallthrough]] attribute, 364
static assertions, 947–948
static binding, 287
static casts, pointers and, 176
static data members, 258
access, 259–260
const, 260–261
inline variables, 259
static keyword
data members, 347
linkage, 347
external, 348
internal, 348
static, 348
methods, 347
static methods, 251
static variables, 350
static_cast( ), 358–359, 361
std::array container – strings

std::array container, 25
std::as_const( ) method, 358
std::atomic type, 827
std::back_inserter, 73
std::bad_cast exception, 360–361
std::condition_variable, 840
std::condition_variable_any, 840
std::function, 612–614
std::make_from_tuple( ), 719
std::make_move_iterator( ), 734
std::make_pair( ), 342
std::ostringstream class, 425
std::pair class, 714–716
std::pair utility class, 342
std::reverse_iterator, 730–731
std::shared_future, 847–848
std::string class, 62–64
std::string_view class, 67–69
std::throw_with_nested( ), 453
std::tie, 717–718
std::unique_ptr pointer, 182–183
std::vector, 352–353
std::vector array container, 25–26
STL (Standard Template Library), 507
str( ) method, 425
streams, 410, 600
bidirectional, 431–432
buffered, 410, 413
cerr, 410
cin, 410
clog, 410
cout, 410
current position, 411, 426
destinations, 411
file streams, 426–427
fgets( ) function and, 411
<fstream> header file, 426–427
fprintf( ) function and, 411
fputs( ) function and, 411
fread( ) function and, 411
fscanf( ) function and, 411
fwrite( ) function and, 411
ios_base::app constant, 427
ios_base::ate constant, 427
ios_base::binary constant, 427
ios_base::in constant, 427
ios_base::out constant, 427
ios_base::trunc constant, 427
flushes, 413
flush-on-access, 430–431
input
>> operator, 417–418
error handling, 418–419
manipulators, 423
iterators
istream_iterator class, 730
ostream_iterator class, 729
linking, 430–431
output, 411–412
<< operator, 412
cout, 412
error handling, 414–415
manipulators, 415–417
seek( ) method, 428–430
sources, 411
Standard Library, 510
string
in-memory stream, 425
sprintf( ) function, 411
sprintf_s( ) function, 411
sscanf( ) function, 411
tell( ) function, 411
tie( ) method, 430–431
unbuffered, 410
string class, 509
<string> header, 509
string literals, localizing, 664
string streams
in-memory stream, 425
sprintf( ) function, 411
sprintf_s( ) function, 411
sscanf( ) function, 411
strings, 29, 598–599
dynamic, 58
C-style, 58–59
nonstandard strings, 69
std::string class, 62–67
std::string_view class, 67–69
string literals, 60–62
interviews, 1040–1041
time point conversion, 69
string_view class, 50
structured bindings, 26
structures, pointers to, 31
subscripting operator, overloading, 486–489
operator[ ], 489–490
subsystems, interface and, 140
superclasses, 130, 278
swap( ) algorithm, 647
swap( ) method, 247–248, 546
swap and exchange algorithms, 646–647
switch statements, 18–19
[[fallthrough]] attribute, 364
symbol servers, 946–947
system tests, 929
system_clock class, 698

tearing, multithreaded programming and, 817
tell( ) method, 428–430
template classes, writing, 977–979
templates, 102, 374, 514
class templates
< > (angle brackets), 383
alias templates, 399
compiler and, 383–384
definitions, 384–386
driving from, 397–398
inheritance, 397–398
instantiation, 378, 382
parameters, 386–391
writing, 375–383
function templates, 400–401
class templates and, 403–404
overloading, 402–403
return types, 405–406
specialization, 401–402
generic programming and, 508
interviews, 1052–1053
metaprogramming, 797–798
factorial at compile time, 798–799
loops, unrolling, 799–800
tuples, printing, 800–803
type traits, 803–812
parameterization
parameter deduction, 404–405
types, 374
values, 374
parameters, non-type, 780–782
recursion, 787
N-Dimensional grid, 788–792
resources, 1067
reusable code, 148
benefits, 149
versus inheritance, 150
problems, 149
std::vector inheritance, 148
specialization, 395–397
full class, 782–785
partial class, 782–785
Standard Library, 508–509
template parameters, 778–780
type parameters, 776–778
variable templates, 407
variadic
folding expressions, 796–797
mixin classes, 795–796
type-safe variable-length argument lists, 792–794
temporary objects, object handling and, 886–887
terminate( ) function, 444
ternary operators, 13
testing
black-box, 909
bugs, 910–913
granularity, 916
integration, 929
JSON-based file serializer, 928
shared resources, readers/writers, 928
wrapper, third-party library, 928–929
interviews, 105
regression testing, 930
resources, 1071
smoke testing, 930
system tests, 929
unit testing, 913–914
approaches, 914–915
building, 922–923
debugging, 927
granularity, 915–916
negative tests, 923
planning, 917
results, 927
running, 919, 922–923
sample data, 918
Visual C++ testing framework, 920–921
writing, 918–919, 921–922
white-box, 909
tgamma( ) function, 511
this pointer, 206–207, 227
<thread> folder, 513, 819
thread objects, 819
threadFunc( ) function, 825
threading design, 853–855
threading support library, header files, 1082
threads
cancelling, 824
exceptions
  copying, 824–826
  rethrowing, 824–826
function object and, 820–822
function pointer and, 819–820
lambda expressions and, 822
local storage and, 823
member function and, 823
results, retrieving, 824
throw keyword, 454–456
throwing exceptions, 437–439, 976
  multiple, 441–443
tie( ) method, 430–431
time points, 699–702
time utilities, Standard Library, 512
timed mutex classes, 832–833
time_point class, 698, 700–702
transform( ) algorithm, 526
transform_exclusive_scan( ) algorithm, 657
transform_inclusive_scan( ) algorithm, 657
transparent operator functors, 622
try keyword, 464–467
tuples
  comparisons, 718–719
  concatenation, 718
decompose
  std::tie, 717–718
  structured bindings, 717
std::apply, 719–720
std::make_from_tuple( ), 719
std::pair class, 714–716
std::tuple_cat( ), 718
<typename> header, 714–716
type aliases, 352–353
  function pointers, 353–355
  pointers
    to data members, 355–356
    to methods, 355–356
type building, operator overload and, 271–272
type inference, 38
  auto keyword, 39–40
dcltype keyword, 40
type parameters, templates, 776–778
typed constructors, 307
typedefs, 356–357
types
  enumerated, 15–16, 264–265
    strongly typed enumerations, 16
  literal, 346
  structs, 16–17
  templates and, 384
<typename_traits> header, 513

U

UCS (Universal Character Set), 665
UML (Unified Modeling Language), 96, 1067
unary operators, 13
unbuffered streams, 410
uncaught exceptions, 444–445
Unicode, 509
uniform initialization, 43–45
unique() algorithm, 526
unique_copy() algorithm, 526
unique_ptr pointer, 182–186
unit testing, 913–914
approaches, 914–915
building, 922–923
debugging, 927
granularity, 915–916
negative tests, 923
planning, 917
results, 927
running, 919, 922–923
sample data, 918
Visual C++ testing framework, 920–921
writing, 918–919, 921–922
unordered_map container, 521
unordered_multimap container, 521
unordered_multiset container, 521
upcasting, 299
upper_bound() algorithm, 649
user-defined literals, 365–367
using declaration, 9
utilities
  general, header files, 1079–1080
  interface, 140
  mathematical, header files, 1081
  variable-length argument lists, 369–370
    argument access, 370–371
    disadvantages, 371
<vutility> header, 630
tuples, 714–716

V

value semantics, elements, 537–538
variable templates, 407
variables, 10–12
  automatic variables, 164
casting, 12
casted, 12
condition variables, 840–842

W

const keyword, 343
inline, static data member access, 259
methodPtr, 355–356
non-local
destruction, 351
initialization order, 351
static, 350
uninitialized, 30

variadic templates
  folding expressions, 796–797
  mixin classes, 795–796
  parameter packs, 793
type-safe variable-length argument lists,
  792–794
<variant> header, 712–713

vector container, 342, 514–515, 519, 542–561
algorithmic complexity, 554
allocators, 542
assigning, 545–546
comparing, 546
constructors, 544–545
copying, 545–546
destuctors, 544–545
dynamic-length, 544
elements, 550–553
fixed-length, 542–543
iterators, 547–550, 554
memory allocation scheme, 554–556
move semantics, 553–554
reference storage, 550
round-robin schedule example, 556–561
vector_bool specialization, 561
vendor-specific attributes, 365
virtual base classes, 312, 331–332
virtual keyword, 281–282
destuctors, 288–290
implementing, 287–288
virtual methods, 303
Visual C++ testing framework, 920–921
vtables, 287

Waterfall Model life cycle model, 861–863
weak_ptr, 189–190
Weather Prediction class, 291–294
while loop, 26–27
white-box testing, 909
wide characters, localization, 664–665
wfstream, 665
Windows, DLL (Dynamic Link Library), 354–355
wofstream, 665
write( ) method, 413

Wrox downloads, 3
ws input manipulator, 423

X-Y-Z
xRef, 334–335

zero-initialization constructs, 544