Index

Symbols

– (subtraction symbol), 60
–– (minus minus)
  decrement operator, 84, 92
  prefix/postfix operators, 85
  switch prefix, 166
! (bang), 92
!= (not equal to) operator, 107–108
(pound), 92, 96
% (percent sign) modulus operator, 86
& (ampersand), 92
  reference operator, 197, 220, 496, 501, 505, 510, 647
&& (and) operator, 110–111
( ) (parentheses), 65–66, 92, 202
* (star; asterisk), 92, 501
  dereference operator, 201
  multiplication symbol, 60, 84
  pointer variables, 205, 220
. (dot) operator, 92, 250, 555
. . . (ellipsis)
  general exception catcher, 567
  variadic operator, 422
/ (forward slash), 92
  division symbol, 60, 86
  in pathnames, 688, 729
  switch prefix, 166
/* */ (delimiters), 271
// (double slash), 270–271
: (colon)
  conditional operators and comparisons, 99
  range operator, 128
:: (two colons)
  classes and functions, 337, 575, 583, 593
  scope resolution operator, 238
; (semicolon), 59, 153, 234
? (question mark), 99
@ (at sign), 92
[ ] (square brackets), 92, 388
\ (backslash), 66, 90–92, 688, 734
^ (caret), 93
{ } (curly braces), 53, 55, 92–93
|| (or) operator, 110–111
~ (tilde), 92
+ (addition symbol), 60, 78, 97
++ (plus plus)
  increment operator, 81, 92
  prefix/postfix operators, 85
++i (pre-increment) operator, 126
+= (plus equal) notation, 81, 96
< (less than)
  less than operator, 108–109
  operator function, 653–654, 668
< > (angle brackets), 93, 181–182
<< (insertion) operator, 59–60, 102, 223, 300, 661, 672, 698–699, 726, 738–741
<= (less than or equal to) operator, 108
= (equals sign), 75, 417–418
= 0 (pure specifier), 333–334
== (equal equal), 92
  equal to operator, 108–109
<=> (spaceship) operator, 109–110, 216
> (greater than) operator, 108–109
–> (arrow) operator, 245, 247, 250, 555, 608
>= (greater than or equal to) operator, 108
>> (extraction) operator, 102, 223, 277, 284, 712–714, 738, 741
" " (double quotes), 59, 67, 90, 93, 181–182
' ' (single quotes), 88, 90
\0 (null) character, 89

A

absolute paths, 846, 853
abstract classes, 333–334
abstract virtual methods, 334
Acronym Finder, 48
AddFiles example, 170
AddInteger example, 78–79
AddInteger2 example, 79–81
addition symbol (+), 60, 78, 97
AddOne example, 149
AddOne( ) function, 149–153
AIDE (Android IDE), 33–34
Alexander, Christopher, 336
algorithms, Standard Library, 755–757
allocating/deallocating memory
  defined, 205
  using new operator, 208
American National Standards Institute (ANSI), 515–516
ampersand (&), 92
  reference operator, 197, 220, 496, 501, 505, 510, 647
and (&&) operator, 110–111
Android IDE (AIDE), 33–34
AndroidForums, 42
angle brackets (< >), 93, 181–182
anonymous functions, 398

COPYRIGHTED MATERIAL

ANSI (American National Standards Institute), 515–516
APIs (Application Programming Interfaces), 316
application flow, 105–138
  conditions, 107–115
    defined, 105
    evaluating, 107–111
    evaluating multiple, 110–111
    evaluating with `if` statements, 111–115
    in `for` loops, 118–122
    operators, 108–110
    satisfying, 107
  decisions, 106–107
  `if` statements, 111–115
    defined, 105
    with `else` keyword, 112–115
    example of, 111–112
  loops, 115–138
    breaking, 133–135
    continuing, 133, 135–136
    defined, 105–106
    `do–while` loops, 116–117, 132–133
    `for` loops, 116–130
    nesting, 136–138
    `while` loops, 116–117, 130–132
Application Programming Interfaces (APIs), 316
arguments. See parameters
`Array01` example, 484–485
`Array02` example, 486
`Array03` example, 487
`Array04` example, 488
`Array05` example, 498
`Array06` example, 498
`ArrayLoop` example, 303
`ArrayPassing` example, 306–307
`ArrayPointer` example, 304–305
arrays, 302–308, 482–498
  accessing, 302–304
  adding and subtracting pointers, 307–308
  allocating on heap, 494
  array names, 484, 486
  arrays of arrays, 495–498
  arrays of pointers, 304–305, 495–498
  command-line parameters and, 492–494
  constant, 498
  declaring, 303–304, 482–484
  defined, 167, 208, 302
  deleting from heap, 494–495
  external declarations, 485–487
  inner and outer, 496
  members, 496
  multidimensional arrays, 488–492
  overcoming limitations of, 642–643
  passing pointers to, 484–485
  passing to functions, 306–307
  pointer types, 487–488
  sorting, 494–495
  string arrays using pointers, 503–504
arrow (`->`) operator, 245, 247, 250, 555, 608
ASCII table, 88
assembly code, 452–454
  defined, 473
  tracing through, 475
`AssignLambda` example, 420–421
assignments, 108
asterisk (`*`), 92, 501
  dereference operator, 201
  multiplication symbol, 60, 84
  pointer variables, 205, 220
at sign (`@`), 92
atomic operations, 757–759
`attr` element, 400
`Auto` example, 388–389
`auto` keyword, 388–390
  lambda expressions, 404–405
  pointing to functions, 506
automata programming, 312
automatic type determination
  functions, 146
  lambda expressions, 401–404
  `for` loops, 129–130

B

backslash (`\`), 66, 90–92, 688, 734
bang (`!`), 92
`BasicString` example, 786
`begin()` function, 388, 655
`BestFriends` example, 589–590
Bitcoin, 820
`body` element, 401
`boolalpha` flag, 702–703
Boole, George, 100
Boolean expressions, 101
Boolean variables, 100–101
  defined, 100
  size and range, 516
  using with conditional operators, 101
Boost, 817–868
  AutoIndex utility, 834
  bcp (Boost copy) utility, 833, 843–845
  `BOOST_FOREACH` macro, 862–864
  `BOOST_REVERSE_FOREACH` macro, 863
  BoostBook format, 833, 840
  Boost.Build, 833–837
  building date/time application, 846–848
  building libraries, 825–827
  compiler support, 822
  downloading, 821, 823

features backported to Standard Library, 818–819
features of, 821
Filesystem library, 864–868
folders, 824
header-only libraries, 825
Inspect utility, 833, 837–840
installing, 823–824
installing on 64-bit systems, 828, 852
licensing, 822
numeric conversion, 858–861
paid support, 823
QuickBook add-on, 833, 841–843
RegEx library, 850–857
smart pointers, 191
Standard Library vs., 819–820
testing, 827, 829–833
tokenizer, 857–858
tools, 833–835
Wave utility, 834, 845
BoostPro Computing, 823
`break` statements, 133–135, 137, 296, 299
breakpoints, 454, 457–467
defined, 457–458
enabling/disabling, 460–463
inspecting variables, 463–467
setting in Code::Blocks, 459–460
`Breakpoints` example, 458–450
`Breakpoints2` example, 463–464
`BuggyProgram` example, 444
bugs, 429–442
anticipating and avoiding, 432–442
creating objects, 441–442
forgetting deleted objects, 441
indexes, 441
menus, 432–435
`new` and `delete`, 441
string processing, 437–440
textual input, 435–436
debugging, 443–456
Code::Blocks debugger, 444–455
command-line arguments, 456
Dev-C++ debugger, 456
gdb debugger, 456
programming with debuggers, 444–453
stacks, 469–478
Visual C++ debugger, 455
features that resemble bugs, 430–432
myth of bulletproof applications, 440
origin of term, 429
built-in literals, 780–781

C

C compatibility headers, 759
C++ 20, 269–308
application flow, 105–138
arrays, 302–308, 482–498
bugs, 429–442
characteristics of, 1
`cin` object, 277–281
classes, 227–268, 571–599
code editing, 53–55
comments, 270–272
compiling (building), 57, 67–68
conditional operators, 98–101
constants, 292–295
constructors, 542–563
design patterns, 335–366
desktop configuration, 9–26
destructors, 548–550, 558–563
directories, 727–736
exceptions, 563–569
executing (running), 52–53, 67–68
functional programming, 312, 369–395
functions, 139–168
linking, 67
mobile configuration, 27–43
namespaces, 534–539
objects, 309–334
pointers, 187–223, 498–510
popularity of, 45
preprocessor directives, 282–291
random numbers, 300–302
references, 510–513
saving code, 62
source code files, 169–186
Standard Library, 637–680, 753–778
streams, 683–726
strings, 93–98
structures, 515–539
`switch` statements, 295–298
templates, 795–815
type conversion, 272–277
User-Defined Literals, 779–794
variables, 69–91
wrapping `enum` types, 298–299
C++ Shell IDE, 35
C4Droid IDE, 29, 33
`captures` element, 400
caret (^), 93
`CarParts` example, 354–357
carriage return (`\r`) character, 90–91, 281
casting data, 520–529
`const_cast`, 528–529
converting data vs., 521–522
`dynamic_cast`, 524–527
narrowing casts, 523
`static_cast`, 527
`CastOrConvert` example, 521

catch blocks
  catching any exception, 567–568
  multiple, 565–566
  try...catch blocks, 558, 563–565
.cbp files, 20
ChangePointer example, 203–204
ChangeVariable example, 74
character variables, 88–91
  carriage return character, 90
  defined, 88
  initializing, 88
  newline character, 89–90
  nonprintable characters, 89
  null character, 89
  size and range, 516
  tab character, 90
  values, 88
Cheesecake example, 597–598
CheeseClass example, 253–254
Church, Alonzo, 398
cin (console input) object, 102–103, 277–281
class definitions, 230
  defined, 234, 241
  separating method code, 238
class keyword, 232
classes, 227–268, 571–599
  accessing members, 241–244
  class definitions, 230
    defined, 234, 241
    separating method code, 238
  const parameters, 251–252
  constructors
    adding parameters to, 263–264
    overview, 259–261
    with stack variables, 262
  creating with templates, 601–636
    creating templates, 605–607
    deriving templates, 623–630
    function templates, 630–636
    including static members in templates, 611–612
    need for, 602–605
    parametering templates, 612–622
    separating templates from function code, 609–611
    template keyword, 607–608
    typedefs for templates, 622–623
    types, 602
  defined, 230, 240
  destructors, 260–262
  friend classes, 588–591
  header files
    class names and, 230
    defined, 241
    properties of, 232
  hierarchies of, 264–267
  implementing, 232–237
  inheritance, 571–591
    access adjustment, 574–575
    avoiding polymorphism, 573–574
    avoiding variable naming conflicts, 575–576
    class-based access adjustment, 575–576
    friend classes and functions, 588–591
    multiple inheritance, 581–584
    overriding functions, 577–580
    polymorphism, 572
    virtual inheritance, 584–588
  instances
    of classes, creating, 234–237
    defined, 230
    of objects, creating, 237
  lambda expressions with, 407–408
  making streamable classes, 737–749
  methods, 232
    defined, 241
    naming, 238–239
    overloading, 256–259
    running, 235–236
    separating code for, 237–240
  modeling, 232
  names and filenames, 230
  nesting, 591–596
  object aliases, 267–268
  objects
    capabilities of, 228–229
    defined, 240
    overview, 227–229
    passing to functions, 249–251
  parts of, 240–241
  persistent classes, 738
  properties, 231
    accessing, 235
    defined, 241
  raw pointers and, 244–248
  singleton, 230
  smart pointers and, 248–249
  source files, 240
    defined, 241
    method code, 230
  structures vs., 530
  this pointer, 252–256
  types within, 597–599
ClassFromTemplate example, 624–625
ClassTemplate example, 804–806
close() method, 689
Code::Blocks IDE, 9–25
  auto indentation, 60–61
  Boost
    building libraries, 825–827
    testing, 827, 829–833
  breakpoints, 454
    enabling/disabling, 460–463
    setting, 459–460

command-line arguments, 168
copy of C++ included with, 9
CppDroid vs., 39–40
creating multiple source code files, 170–177
debugger, 445–455
application stack, 471
assembly language version of code, 452–454
basic functionality, 450–453
breakpoints, 454
call stack, 454, 471–473
CPU registers, 454, 478
current stack frame information, 455
debug and symbol information, 445
Disassembly window, 452–454, 475–476
FPU status, 455
initial run, 446–448
instruction pointer, 447, 450–451
line-by-line code review, 449–450
loaded libraries, 455
memory dump, 454, 496
parts of, 453–455
running threads, 454
signal handling, 455
stepping into functions, 449
targets and files, 455
watches, 454, 465
defined, 11
downloading, 11
error reports, 57–58
features of, 17–25
file associations, 18
installing, 11–17
in Linux, graphical installation, 15–17
in Linux, standard installation, 14–15
in Mac OS/X, 13–14
storage location, 11–12, 19
in Windows, 12–13
libraries, 809–815
configuring, 812–813
defining, 810–812
support for, 809–810
projects, 45–52
application types, 46–49
building and executing applications, 52–53
creating, 46
defined, 46–47
defining, 47–52
elements of, 46
naming, 49–50
storage location, 49–50
recommended version of, 10–11
sample projects source code, 19–20
selecting compiler, 24–25
smart pointers, 212
starting for first time, 18–19
support for application types, 46–49
tip dialog boxes, turning on/off, 18–19
user interface, 21–24
CodeChef IDE, 35
CodeLite IDE, 26
colon (:)
conditional operators and comparisons, 99
range operator, 128
two colons (: :)
classes and functions, 337, 575, 583, 593
scope resolution operator, 238
`Combine` example, 161–162
Command Line Tools for Xcode, 14
Command Prompt, 167
command-line arguments, 166–168
in Code:Blocks, 168
debugging with command-line arguments, 456
defined, 166
command-line parameters
accessing, 167
arrays and, 492–494
`CommandLineParameters` example, 167
`CommandLineParams` example, 492–493
comments, 270–272
delimiters, 271
double slash, 270–271
utility of, 270
compilers, 9–10
defined, 10
IDEs vs., 10
selected at Code::Blocks IDE install, 13
compiling (building), 63
defined, 57, 67
shortcuts for, 68
complex numbers, 788
`ComplexNumber` example, 788–789
`CompoundData` example, 533
concatenating strings, 96–97
concepts, Standard Library, 759–760
concurrency, 767
conditional expressions, 99
conditional operators, 98–101
defined, 98
using with Boolean variables, 101

conditions, 107–115
defined, 105
evaluating, 107–111
evaluating multiple, 110–111
evaluating with `if` statements, 111–115
in `for` loops, 118–122
operators, 108–110
satisfying, 107
`ConnectNames` example, 154–155
Console Application, 48
console input (`cin`) object, 102–103, 277–281
console output (`cout`) object, 58, 72–73, 300
`const` parameters, 251–252, 292
constant casts (`const_cast`), 527
constant expressions (`constexpr`), 378–380
`ConstantExpression` example, 379
constants, 251–252, 292–295
accessing shortcuts to math constants, 294–295
creating, 292–293
creating with `#define` directive, 283, 285–286
`Constants` example, 292–293
`ConstCast` example, 528–529
`consteval` specifier, 416–417
`constexpr` (constant expressions), 378–380
`Constructor01` example, 542–543
`Constructor02` example, 546–547
`Constructor03` example, 547
`Constructor04` example, 549
`Constructor05` example, 551
`Constructor06` example, 552
`Constructor07` example, 552–553
`Constructor08` example, 555–556
constructors, 542–563
adding and removing items in, 347–349
adding parameters to, 263–264
calling one constructor from another, 553–554
copy constructors, 555–557
defined, 542
errors, 557–558
functional constructors, 550–553
as initialization functions, 260
initializing members, 542–548
accessing base constructors, 545–546
adding initializers to classes, 544
default values, 547–548
overloading constructor, 546–547
passing variables, 544–545
ordering, 562
overloading, 542–543, 546–547
overview, 259–261
with stack variables, 262
container adapters, 419
container classes, 638–674
comparing instances, 649–654
copying, 648–649, 673–674
defined, 638
double-ended queues, 669–670
fixed-size arrays, 642–643
intersections, 662–664
iterators, 655–658
lists, 664–669
mapping data, 643–644
`Pair` template class, 658
pointers, 644–649
queues, 670–672
references, 644–648
sets, 658–664
stacks, 670–672
Standard Library, 760
storing instances, 644–649
streams and, 692–693
unions, 662–664
vectors, 639–642
`continue` statements, 133, 135–137
control characters, 88, 784
conversion UDLs, 792
`Convert` example, 858–860
converting data
built-in conversions, 522
casting data vs., 521–522
cooked literals, 784–785
`CoolHolder` example, 605–607
`CopiedFiles` example, 173–174
`copy` command, 734
copy constructors, 555–557
`CopyContainer` example, 673–674
`CopyVariable` example, 75
coroutines, 760
`cout` (console output) object, 58, 72–73, 300
`.cpp` files, 20, 638
CppDroid IDE, 27–32, 35–43
advantages of, 29
alternatives to, 29, 32–35
Code::Blocks vs., 39–40
downloading and installing, 31–32
examples and tutorials, 37, 42–43
features of, 30–31
free vs. paid versions, 28, 30–31
help resources, 40–42
projects, 37–39
accessing samples, 37–38
closing, 39
creating, 39
creating source code, 38–39
deleting, 39

opening existing, 39
saving, 39
user interface, 35–36
CrackingDiamonds example, 584–585
CrackingDiamonds2 example, 587
CreateMacro example, 405–406
CreateString example, 93
curly braces ({ }), 53, 55, 92–93
cursor, 53–54
custom type UDLs, 793
CustomManipulator example, 745–747
CustomUDL01 example, 792
CustomUDL02 example, 793
CxxDroid IDE, 29

D

data packs, 422
debug versions, 286–287
debugging, 443–456
Code::Blocks debugger, 444–455
application stack, 471
basic functionality, 450–453
debug and symbol information, 445
initial run, 446–448
line-by-line code review, 449–450
parts of, 453–455
command-line arguments, 456
debuggers, defined, 444
Dev-C++ debugger, 446, 456
gdb debugger, 446, 456
selecting at Code::Blocks install, 13
stacks, 469–478
application stack, 470
defined, 470
popping data off of, 470
pushing data onto, 470
registers, 470
stack frames, 473, 476–477
stack pointer, 470
storing local variables, 473–475
tracing through assembly code, 475–478
viewing threads, 475
Visual C++ debugger, 455
dec flag, 702–703
declarations
defined, 382
functions vs., 383–384
removing side effects, 385–387
side effects of, 383–384
Declarative example, 387
declarative programming, 370, 373, 387–388
decltype() operator, 415, 418
deep copies, 250
default constructors, 548–550
#define directive, 283, 285–286, 294
delete operator, 209, 441, 676–677
delimiters, 94, 271
deque (double-ended queue) class, 669–670
DereferencePointer example, 199
DerivingTwoDiff example, 581
DerivingTwoDiff2 example, 583
deserialization, 738
design patterns, 335–366
defined, 335
Façade pattern, 353
mediator pattern, 349–366
creating example, 354–366
outline of example, 351–354
overview, 350–351
observer pattern, 341–349
automatically adding observers, 347–349
creating observer pattern classes, 343–345
overview, 341–343
Standard Library and, 346–347
origin of, 336
reusability and, 336
singleton pattern, 337–340
Design Patterns (Gamma, Helm, Johnson, and Vlissides), 336
desktop configuration for C++, 9–26
compilers, 9–11
IDEs, 9–26
alternatives to Code::Blocks, 25–26
downloading Code::Blocks, 11
features of Code::Blocks, 17–25
installing Code::Blocks, 12–17
installing C++, 9–10
Destructor01 example, 558–559
Destructor02 example, 560
Destructor03 example, 561
destructors, 558–563
adding and removing items in, 347–349
adding default, 548–550
defined, 542
destroying instances, 558–560
as finalization functions, 260
ordering, 562
overview, 260–261
with stack variables, 262
virtually inheriting, 560–563
Dev-C++ debugger, 446, 456
Dev-C++ IDE, 26
diamond-shaped inheritance problem, 584–588

directories, 727–736
copying files, 733–734
creating, 728–730
deleting, 730
getting contents of, 731–733
moving, 736
moving files, 736
renaming, 735–736
renaming files, 735–736
DirectoryCheck01 example, 694
DirectX, 29
DisplayEnum example, 298–299
DivideInteger example, 87–88
DLLs (dynamic link libraries), 49, 810
do...while loops, 720
DocBook Document Type Definition (DTD), 841
DocBook eXtensible Stylesheet Language (XSL), 841
done variable, 131–132
DoorClass example, 256–258
dot (.) operator, 92, 250, 555
double keyword, 516
double quotes (" "), 59, 67, 90, 93, 181–182
double slash (//), 270–271
double-ended queue (deque) class, 669–670
do–while loops, 132–133
defined, 116–117
when to use, 117
DoWhileLoop example, 132–133
DTD (DocBook Document Type Definition), 841
Duration example, 790
dynamic allocation, using raw pointers for, 188
dynamic arrays, 675–677
creating, 675–676
deleting, 676–677
dynamic casts, 524–527
dynamic link libraries (DLLs), 49, 810
dynamic memory management, 205
DynamicArray example, 675–676
DynamicCast example, 524–525

E

–E switch, 283
Eclipse IDE, 26
#elif directive, 290
ellipsis (...)
general exception catcher, 567
variadic operator, 422
#else directive, 287
Emacs IDE, 26
emplace() function, 677, 679
encapsulation, 316–322
APIs, 316
process for, 318–319
properties
implementing, 319–322
overview of, 316–317
private vs. protected, 318
end() function, 388, 655–656
end of file (EOF) condition, 715–720
EOF check approach, 718–720
record count approach, 715–718
#endif directive, 290
endl manipulator, 738, 743–744
enumerations (enums)
naming types, 232
overview, 229
wrapping enum types, 298–299
EOF. See end of file condition
eof() function, 720
equal equal (==), 92
equal to operator, 108–109
EqualityCheck example, 99–100
equals sign (=), 75, 417–418
EquateStruct example, 531–532
erase() function, 164–165, 677
escape-sequences, 67
escaping characters, 688
event-driven programming, 312
exception element, 400
Exception01 example, 563–564
Exception02 example, 569
Exception03 example, 570
exceptions, 322, 563–569
catching any exception, 567–568
defined, 563
dynamic casts, 527
multiple catch blocks, 565–566
rethrowing exceptions, 568–569
specifying that lambda expressions throw, 413–414
standard category exceptions, 569
throwing direct instances, 566–567
try...catch blocks, 563–565
executable files
creating, 52
defined, 67
executing (running), 52–53, 67–68
shortcuts for, 68
stopping execution, 53
execution policy, 757
extraction (>>) operator, 102, 223, 277, 284, 712–714, 738, 741

F

fabs() function, 143–147
Fabs2 example, 145
Fabs3 example, 146–147
Façade pattern, 353

factorial() function, 379–380
fail() method, 694
__FILE__ macro, 288
FileLineCount example, 381–382
FileOutput01 example, 689
FileOutput02 example, 696
FileRead01 example, 690–691, 712–713
FileRead02 example, 715–717
FileRead03 example, 718–719
FileReadWrite01 example, 691
filesystem library, 729
<filesystem> header, 761
FileWrite01 example, 698
FileWrite02 example, 700–701
find() function, 678, 749, 774–776
_findclose() function, 731
_findfirst() function, 731
_findnext() function, 731
FindString example, 774
FirstFunction example, 142
fixed flag, 702–703
floating-point numbers, 70
  fabs() function, 143–147
  size and range, 516
  type conversion, 276–277
flush() function, 692
for loops
  automatic type determination, 129–130
  changing conditions, 119–122
  conditions in, 118–122
  counting backward, 122–123
  declaring counter variable within, 120
  defined, 116–117
  finalizers, 118
  initializers, 118
  multiple initialization variables, 123–126
  placing conditions within declarations, 128–129
  ranges, 126–128
  simple, 118–119
  when to use, 117
for_each() loops, 408
ForCountdown example, 122–123
ForLoop example, 119
ForLoop2 example, 121
ForLoop3 example, 134
ForLoop4 example, 135–136
ForLoop5 example, 136–137
ForLoop6 example, 137–138
ForLoopComplex example, 125
ForLoopCondition example, 128
ForLoopCondition2 example, 129
ForLoopMultiVariable example, 123–126
ForLoopRange example, 127
forward references (function prototypes), 159–161, 177–178, 237
forward slash (/), 92
  division symbol, 60, 86
  in pathnames, 688, 729
  switch prefix, 166
frankenfunctions, 399
free software vs. paid software, 28
FreePointer example, 209–210
friend classes and functions, 588–591
friend keyword, 588
front() function, 670
fstream class, 687, 691
func parameter, 508–509, 744
function literals, 398
function prototypes (forward references), 159–161, 177–178, 237
function templates, 630–636
  defined, 630
  overloading and, 632–635
  templatizing methods, 635–636
functional constructors, 550–553
functional programming, 312, 369–395
  auto keyword, 388–390
  declarations and functions, 382–388
  first-class and higher-order functions, 373
  immutable data, 370–371, 375–380
    constant expressions, 378–380
    immutability in classes and structures, 377–378
    immutable variables, 375–377
  lambda calculus, 373
  lambda expressions, 394–425
    advantages of, 398–399
    assignable stateless, 420–421
    automatic type determination, 401–404
    capture clause, 408–411
    captures with = and this, 417–418
    container adapters, 419
    decltype() operator, 415, 418
    defined, 394
    elements of, 399–401
    full version, 399
    immediate functions, 416–417
    origin of, 398
    pack expansions, 422–425
    patterns for creating, 401
    performing transforms using, 394–395

function templates *(continued)*
sorting data, 411–413
specifying to throw exceptions, 413–414
unevaluated contexts, 418–419
using as macros, 405–406
using `auto` keyword with, 404–405
using with classes and structures, 407–408
variadic templates, 416, 422–425
limitations of C++, 374–375
multiprocessing, 371, 375–376
other paradigms vs., 370–373
passing functions to functions, 390–394
pure functions, 371, 374
recursion, 374–375, 379
referential transparency, 374
state, 370–371, 381–382
`FunctionFunction` example, 391–392
`FunctionOverloadingAndTemplates` example, 632
`FunctionPointer01` example, 505
`FunctionPointer02` example, 508
`FunctionPointer03` example, 509–510
functions, 139–168
anonymous functions, 398
automatic type determination, 146
calling, 144–145
declarations vs., 383–384
defined, 139, 141
defining custom functions, 148–153
division of work, 139–145
forward references, 159–161
frankenfunctions, 399
friend functions, 588–591
inner workings of, 150–151
local variables, 157–159
as machines, 141–142
`main()` function, 165–168
naming, 150, 152
nested calls, 471–472
operator, 653–654
overriding, 577–580
parameters, 148, 151–156
passing arrays to, 306–307
passing functions to functions, 390–394
passing multiple variables, 147–148
passing objects to, 249–251
passing pointer variables to, 218–221
passing to by reference, 221
passing to by value, 221
passing values of variables to, 146–147
passing values to, 144–145
pointing to, 505–506
public and private, 242–244
return values, 150–152
returning nothing, 156–157
returning pointer variables from, 221–223
returning references from, 511–513
spelling checks with, 143–144
string functions, 163–165
top-down design, 140–141
two versions of same, 161–162

G

Gamma, Erich, 336
Gang of Four, 336
garbage collection, 206
GCC. See GNU Compiler Collection
gdb debugger, 446, 456
`getch()` function and `_getch()` function, 280–281, 296–297
`getcwd()` method, 699
`GetDirectoryContents` example, 731–732
`getline()` function, 724, 726
getters, 316–317
defined, 316
property vs. method approach, 319
reasons for using, 317
`getvolume()` function, 803
`GetWorkingDirectory` example, 700–703
`getx()` method, 610, 621–622
GIMP (GNU Image Manipulation Program) Tool Kit Plus (GTK+) Project, 48
global variables, 183–185
`GlobalVariable` example, 184
GNU Compiler Collection (GCC) version 8.3 compiler, 9–11
configuring to use C++ 17 standard, 126–127
downloading, 11
platform support, 10
selecting and setting as default, 24–25
verifying installation, 14
GNU Image Manipulation Program (GIMP) Tool Kit Plus (GTK+) Project, 48
GoogleCodeExporter, 820
greater than (`>`) operator, 108–109
greater than or equal to (`>=`) operator, 108
Guideline Support Library (GSL), 523

H

hash map, 770
hashes, 768–771
`HashingStrings` example, 769
`HashMap` example, 770
header files, 240
adding header once, 182
class names and, 230

defined, 179, 241
properties of, 232
streams and, 686–687
using angle brackets, 182–183
using quotes, 182–183
header wrappers, 185–186
heaps, 192–194
arrays
allocating on, 494
deleting from, 494–495
defined, 192
dynamic arrays, 675
dynamic memory management, 205
Helm, Richard, 336
hex flag, 702–703
hexadecimal (hex) numbers
converting between decimal numbers and, 195–196
overview, 190
hierarchies of classes, 264–267
creating, 265–266
types of inheritance, 266–267
hierarchies of objects, 322–334
abstract classes, 333–334
establishing, 322–324
overriding methods, 330–332
polymorphism, 332–333
protecting members when inheriting, 324–330

I

i++ (post-increment) operator, 126
Ideone IDE, 35
IDEs. See Code::Blocks IDE; CppDroid IDE; Integrated Development Environments
if statements, 111–115
defined, 105
with else keyword, 112–115
example of, 111–112
#ifdef directive, 287, 290
IfElse example, 112–113
IfElse2 example, 113–114
IfElse3 example, 114–115
ifstream class, 687, 718, 720–721
ImFree example, 609–610
immutable data, 370–371, 375–380
constant expressions, 378–380
immutability in classes and structures, 377–378
immutable variables, 375–377
Immutable example, 376
Immutable2 example, 377–378
imperative programming, 371–372
ImplementProperties example, 319–321
#include directive, 294–295
indentation, 30, 60–61
indexes, 95, 302–303, 441, 482
IndividualCharacter example, 94
IndividualCharacter2 example, 94
inheritance, 571–591
access adjustment, 574–575
avoiding polymorphism, 573–574
avoiding variable naming conflicts, 575–576
class-based access adjustment, 575–576
defined, 265
friend classes and functions, 588–591
multiple inheritance, 581–584
overriding functions, 577–580
polymorphism, 572
protection during, 324–330
types of, 266–267
virtual inheritance, 584–588
InheritedMembers example, 325–328
Initializer example, 208
initializers
adding to classes, 544
creating raw pointers, 206–208
for loops, 118
setting up properties, 544
inner class definitions, 594–596
insert() function, 163, 165, 665, 677–679
insertion (<<) operator, 59–60, 102, 223, 300, 661, 672, 698–699, 726, 738–741
insertion point, 89
integer overflow, 64–65
integer variables, 78–88
adding, 78–82
creating, 70–73
defined, 70
dividing, 86–88
multiplying, 84, 86
size and range, 516
subtracting, 82–84
type conversion, 272–278
Integrated Development Environments (IDEs). See also Code::Blocks IDE; CppDroid IDE
AIDE, 33–34
alternative, 25–26
C++ Shell, 35
C4Droid, 29, 33
CodeChef, 35
CodeLite, 26
compilers vs., 10
CxxDroid, 29
Dev-C++, 26
Eclipse, 26
Emacs, 26
Ideone, 35
JDoodle, 34
Netbeans, 26
OnlineGBD, 35
Qt Creator, 26
RepLit, 35
Rextester.com, 35
TutorialsPoint, 35

internal flag, 702–703
International Standards Organization (ISO), 753–754
Internet connections, 558
invoking constructors, 208
ios class and flags, 689, 695–696, 702–703
is_same() function, 410
ISO (International Standards Organization), 753–754
istream class, 691, 718, 720, 741
IteratorPointer example, 657
iterators
  categories and, 761–762
  containers and, 655–658
  defined, 382
  random access iterator, 771–774
  Standard Library, 761–762
Iterators example, 656

J

JDoodle IDE, 34
Johnson, Ralph, 336

L

lambda abstractions, 398
lambda calculus, 373, 398
Lambda example, 394–395
lambda expressions, 255–256, 394–425
  advantages of, 398–399
  assignable stateless, 420–421
  capture clause, 408–411
  captures with = and this, 417–418
  container adapters, 419
  decltype() operator, 415, 418
  defined, 255, 394
  elements of, 399–401
  full version, 399
  immediate functions, 416–417
  origin of, 398
  pack expansions, 422–425
  patterns for creating, 401
  performing transforms using, 394
  sorting data, 411–413
  specifying to throw exceptions, 413–414
  unevaluated contexts, 418–419
  using as macros, 405–406
  using auto keyword with, 404–405
  using this pointers in, 255–256
  using with classes and structures, 407–408
  variadic templates, 416, 422–425
LambdaForClass example, 407–408
left flag, 702–703
length() function, 202–203
less than (<)
  less than operator, 108–109
  operator function, 653–654, 668
less than or equal to (<=) operator, 108
libraries
  Boost, 817–868
  defined, 818
  Standard Library, 3, 374–375, 386–387, 637–680, 753–778
  static libraries, 809, 813–814
  template libraries, 809–815
line markers, 283
Linux, installing Code::Blocks IDE
  flavor support, 14–15
  graphical installation, 15–17
  lack of Boost support, 15
  standard installation, 14–15
LISP, 398
list class, 664–668
  adding items to, 665
  defined, 664
  vector class vs., 669
Lists example, 665–667
literals. See User-Defined Literals
localization, Standard Library, 763
Logs & Others window, in Code::Blocks IDE, 23–24
long keyword, 516
loops, 115–138
  for_each() loops, 408
  BOOST_FOREACH macro, 862–864
  BOOST_REVERSE_FOREACH macro, 863
  breaking, 133–135
  continuing, 133, 135–136
  declaring and accessing arrays, 302–303
  defined, 105–106
  do–while loops, 132–133, 720
    defined, 116–117
    when to use, 117
  for loops
    automatic type determination, 129–130
    changing conditions, 119–122
    counting backward, 122–123
    declaring counter variable within, 120
    defined, 116–117
    multiple initialization variables, 123–126
    placing conditions within declarations, 128–129
    ranges, 126–128
    simple, 118–119
    when to use, 117
  nesting, 136–138
  while loops, 130–132, 655, 724

conditions in, 130–131
defined, 116–117
done variable, 131–132
when to use, 117
lvalue term, 517–519
LValueAndRValue example, 518–519

M

M_PI constant, 294–295
Mac OS/X
Command Line Tools for Xcode, 14
Gatekeeper error, 14
installing Code::Blocks IDE, 13–14
macros
creating with #define directive, 283, 285–286
predefined, 288
using lambda expressions as, 405–406
main() function, 165–168
command-line arguments, 166–167
defined, 55
Management window, in Code::Blocks IDE, 22–23
Files tab, 23
FSymbols tab, 23
Projects tab, 23
Resources tab, 23
Symbols tab, 23
manipulators, 738
defined, 742–744
writing, 744–749
map class, 579–580
associating objects with, 643–644
key and value, 658
storing in, 645–648
Maps example, 643–644
Maps2 example, 645–646
math, 60–66, 77
adding and subtracting pointers, 308
common symbols, 60
integer overflow, 64–65
integers
adding, 78–82
dividing, 86–88
multiplying, 84, 86
remainder (modulus), 78
subtracting, 82–84
lambda calculus, 373, 398
mathematical constants, 294–295
orders of operation, 63–64
parentheses, 65–66
MathTemplate example, 799–800
MATLAB, 370
mediator pattern, 349–366
creating example, 354–366
outline of example, 351–354
overview, 350–351
MemberFunctionTemplate example, 635–636
memory leaks, pointers and, 190
menus, anticipating and avoiding bugs, 432–435
methods
accessing properties, 317
defined, 241, 309–310
naming, 238–239
overloading, 256–259
overriding, 330–332
pointing variables to, 506–509
separating code for, 237–240
templatizing, 635–636
minus minus (––)
decrement operator, 84, 92
prefix/postfix operators, 85
switch prefix, 166
mkdir() function, 728–730
mobile configuration for C++, 27–43
AIDE, 33–34
C4Droid, 33
CppDroid, 27–32, 35–43
advantages of, 29
downloading and installing, 31–32
examples and tutorials, 37, 42–43
features of, 30–31
free vs. paid versions, 28, 30–31
help resources, 40–42
projects, 37–39
user interface, 35–36
web-based IDEs, 34–35
modulus (remainder), 78, 86
monadic operations, 191
move() function, 214
Move() template function, 668
multidimensional arrays, 488–492
bounds, 491
data types, 489
declaring, 488–489
initializing, 490
passing, 490–492
multiple inheritance, 581–584
access adjustment, 582
employing, 581–582
issues with, 582–584
polymorphism with, 588
MultipleParameters example, 616–618
MultipleSourceFiles example, 178
MultipleSourceFiles2 example, 180
multiprocessing, 371, 375–376
MultiTask example, 408–409

N

\n (newline) character, 89–91
Named Return Value Optimization (NRVO), 192
Namespace example, 538
namespaces, 534–539
  creating, 534–535
  creating in many places, 537
  defined, 179
  putting variables in, 537
  using namespace line, 535–537
  using part of, 538–539
nargs() function, 422
narrowing casts, 523
nested loops, 136–138
  break and continue statements inside, 137
  defined, 136
NestedCalls example, 471–472
nesting classes, 591–596
  inner class definitions, 594–596
  issues with, 592–594
Netbeans IDE, 26
new operator, 188, 206–211, 245, 441, 494, 676
newline (\n) character, 89–91
noexcept() function, 414
NonTypeParm example, 619–621
NoSideEffects example, 385–386
not equal to (!=) operator, 107–108
(nothrow), 676
NRVO (Named Return Value Optimization), 192
null (\0) character, 89
NULL, setting pointers to, 211
null pointers, 190, 615
nullable references
  optional pointers for, 191
  raw pointers for, 189
nullopt object, 216–217
numeric conversion, 858–861

O

object aliases, 267–268
  creating, 266–267
  defined, 266
ObjectAlias example, 266–267
object-oriented programming (OOP), 227–268, 312, 316
  arrays, 302–308
  cin object, 277–281
  classes, 227–268
    accessing members, 241–244
    class definitions, 230, 234, 238, 241
    const parameters, 251–252
    constructors, 259–264
    defined, 230, 240
    destructors, 260–262
    header files, 230, 232, 240–241
    hierarchies of, 264–268
    implementing, 232–237
    instances, 230, 234–237
    methods, 232, 235–241, 256–259
    modeling, 232
    names and filenames, 230
    object aliases, 267–268
    objects, 227–229, 240, 249–251
    parts of, 240–241
    properties, 231, 235, 241
    raw pointers and, 244–248
    singleton, 230
    smart pointers and, 248–249
    source files, 230, 240–241
    this pointer, 252–256
  comments, 270–272
  constants, 292–295
  defined, 372
  design patterns, 335–366
    mediator pattern, 349–366
    observer pattern, 341–349
    origin of, 336
    singleton pattern, 337–340
  functional programming vs., 371
  objects, 309–334
    collection objects, 311–313
    defined, 309–310
    encapsulation, 316–322
    hierarchies of, 322–334
    mailbox system example, 310–315
    naming, 311
  passing functions to functions, 371
  preprocessor directives, 282–291
  random numbers, 300–302
  switch statements, 295–298
  type conversion, 272–277
objects, 309–334
  collection objects, 311–313
  defined, 141, 240, 309–310
  encapsulation, 316–322
    APIs, 316
    defined, 316
    process for, 318–319
    properties, 316–322
  hierarchies of, 322–334
    abstract classes, 333–334
    establishing, 322–324
    overriding methods, 330–332
    polymorphism, 332–333
    protecting members when inheriting, 324–330
  mailbox system example, 310–315
    classes and instances, 314
    Mailbox objects, 314–315
    Mailboxes collection object, 311–313

necessity of other objects, 315
methods, defined, 309–310
naming, 311
passing to functions, 249–251
properties, defined, 309–310
using pointers with, 247
observer pattern, 341–349
automatically adding observers, 347–349
creating observer pattern classes, 343–345
overview, 341–343
Standard C++ Library and, 346–347
oct flag, 702–703
ofstream class, 687
omanip class, 743–744
online resources
absolute values, 143
Acronym Finder, 48
AMD 128 bit processor, 859
area and volume equations, 800
ASCII table, 88
author's blog, 5–6, 12–14, 456, 822
author's email, 5
author's website, 5
automata programming, 312
bcp utility, 844
Bitcoin, 820
Boolean variables, 100
Boost, 818–823, 828, 836–837, 849–851, 858, 861
BoostBook format, 840–841
C++ compiler support, 9
C++ creator's web page, 80
Cheat Sheet (companion to book), 5
Code::Blocks IDE, 28
checking compiler in, 13
Linux, 15
Mac OS/X, 13
resetting configuration, 13
Windows, 11
companion files for book, 5
complex numbers, 788
concepts, 759
concurrency, 767
converter template, 861
coroutines, 760
CppDroid IDE
AndroidForums, 42
Reddit, 41
SourceForge, 41
StackOverflow, 42
debugging
with command-line arguments, 456
techniques for, 443
design patterns, 336
DTD, 841
event-driven programming, 312
exception catching by value, 565
exception categories, 565
extending structured bindings to be more like variable declarations, 418
filesystem library, 729
<filesystem> header, 761
for_each algorithm, 862
function pointers, 744
GoogleCodeExporter, 820
GTK+ Project, 48
Guideline Support Library, 523
hexadecimal conversion, 195
IDEs
AIDE, 33
C++ Shell, 35
Code::Blocks, 11
CodeChef, 35
CodeLite, 26
CppDroid vs. C4Droid, 33
Dev-C++, 26
Eclipse, 26
Emacs, 26
Ideone, 35
JDoodle, 34
Netbeans, 26
OnlineGBD, 35
Qt Creator, 26
RepLit, 35
Rextester.com, 35
TutorialsPoint, 35
Inspect utility, 840
Kipling, 857
lambda calculus, 373
lambda expressions, 256
libraries, 818
Mac Gatekeeper error, 14
machine learning and deep learning, 369
mathematical constants, 294
monadic operations, 191
Named Return Value Optimization, 192
null pointers, 615
numeric conversion, 861
numeric header, 387
numerics library, 295
object lifetimes and pointers, 256
obvious ideas, 799
OpenGL, 29
optional object, 216
pattern language, 336
predefined macros, 288
programming paradigms, 372
property approach, 319
Qt, 29
QuickBook add-on, 841
ranges, 679
range-v3 library, 764
reactive programming, 312
RegEx library, 851
registers, 470
Return Value Optimization, 192

online resources *(continued)*
ROOT, 30
shallow vs. deep copying, 250
Simple DirectMedia Layer, 29
spaces in paths, 12
Standard Library, 752–753, 785
Standard Library Extensions, 755
`std::array` class, 482
templates, 796
TIOBE Index, 45
tokenizer, 858
Tronic, 820
Unified Modeling Language, 324
`unique_ptr` pointer, 216
Unity, 29
Unreal, 29
updates to book, 5
User-Defined Literals, 780, 785
Visual Studio Code, 28
`volatile` keyword, 528
Waka Waka Bang Splat poem, 93
Wandbox, 415, 679, 729
Wave utility, 845
Windows Calculator, 195
Wine, 29
wxSmith tutorial, 23
XSL, 841
OnlineGBD IDE, 35
OOP. See object-oriented programming
OpenGL, 29
operands, 79
`OperatingOnStrings` example, 164–165
operator functions, 653
operator overloading, 300
operators, 79
`Optional` example, 216–217
`optional` object, 216–217
or (`||`) operator, 110–111
orders of operation, 63–64
`ostream` class, 691, 694
output streams, 697–710
creating fields, 707–710
flags, 700–703
insertion operator, 698–699
setting width, 700, 707–710
specifying precisions, 700, 704–707
`OutputVector` example, 692–693
`OvenClass` example, 242–243
`OverloadedFunctionTemplate` example, 633–634
overloading
constructors, 542–543, 546–547
functions, 632–635
methods, 256–259
operators, 300
overriding methods, 330–332
`OverridingDerived` example, 578–579

P

parameters (arguments), 148, 151–152
adding to constructors, 263–264
defined, 151
functions with multiple, 153–156
functions with none, 153–154
`params` element, 400
parentheses (( )), 65–66, 92, 202
Pattern Language, A (Alexander), 336
`pause` command, 734
`PenClass` example, 232–234
`PenClass2` example, 239–240, 246
`PenClass3` example, 246
`PenClass4` example, 252
percent sign (`%`) modulus operator, 86
persistent classes, 738
plus equal (`+=`) notation, 81, 96
plus plus (`++`)
increment operator, 81, 92
prefix/postfix operators, 85
pointer variables
accessing values through, 199–200
changing variables using, 198–199
declaring, 198
declaring two of same type, 204–205
defined, 197
passing to functions, 218–221
pointing to parts of strings, 202
pointing to something else and back again, 203–204
pointing to strings, 200–203
returning from functions, 221–223
type of, 198
`Pointer01` example, 499
`Pointer02` example, 500
`Pointer03` example, 502
`Pointer04` example, 503–504
`PointerArithmetic` example, 308
pointers, 187–223, 498–510
adding and subtracting, 307–308
arrays of, 304–305, 495–498
changes to in C++ 20, 188–192
dereferencing, 189, 199, 206
external declarations, 485–487
finding variable addresses, 196–198
heaping variables, 192–194
null, 190, 615
passing to arrays, 484–485
pointer variables

accessing values through, 199–200
changing variables using, 198–199
declaring, 198
declaring two of the same type, 204–205
defined, 197
passing to functions, 218–221
pointing to parts of strings, 202
pointing to something else and back again, 203–204
pointing to strings, 200–203
returning from functions, 221–223
type of, 198
pointing to functions, 505–506
pointing to pointers, 500–501
pointing to static methods, 509–510
pointing variables to methods, 506–509
problems with, 190–191
raw
creating using initializers, 206–208
creating using `new` keyword, 206–208
defined, 188
freeing (deleting), 209–211
problems with, 189–191
uses for, 188–189
smart, 191, 211–217
creating, 212–216
defining nullable values using, 216–217
stacking variables, 192–194, 196
string arrays, 503–504
type differentiation, 487–488
`typedefs`
avoiding, 501–502
using, 499–500
using multiple, 502–503
polling, 342
polymorphism, 332–333, 572
avoiding, 573–574
multiple inheritance, 588
`pop_front()` function, 670
post-increment (`i++`) operator, 126
pound (`#`), 92, 96
`pow()` function, 147–148
`Pow1` example, 148
powers of ten, 147–148
precision, 700, 704–707
`precision()` function, 704–707
`PrecisionFunction` example, 705
prefixes
prefix operator, 85
User-Defined Literals, 781–784
`PrefixesAndSuffixes` example, 782–783
pre-increment (`++i`) operator, 126
preprocessor directives, 186, 282–291
basic, 288–291
conditional compilation, 286–287
`#define` directive, 283, 285–286
inner workings of, 283–284
overview, 282–283
`Preprocessor` example, 284
`Preprocessor2` example, 288–289
PriceController example, 613–614
`PrintName()` function, 157–160
`PrintName` example, 157–158
`PrintName2` example, 158–159
`PrintName3` example, 160
`PriorityQueue` example, 418–419
private constructors, 337, 339
private derived classes, 324–325, 329–330
private destructors, 339
`private` keyword, 242
private members, 316, 318, 590–591
procedural programming, 312, 372
programming paradigms, 369, 372–373. See also *names of specific paradigms*
projects, defined, 169
properties
defined, 241, 309–310
implementing, 319–322
private vs. protected, 318
read-only, read/write, and write-only properties, 317, 319–321
protected constructors, 337
protected derived classes, 324–325, 329, 574, 576–577
protected members, 266–267, 316, 318, 590–591
`ProtectingEmbedded` example, 595–596
protocols, defined, 714
pseudorandom numbers, 300
public derived classes, 324–325, 328, 574–577, 582
`public` keyword, 233, 242
public members, 316
pure functions, 371, 374
creating, 375
defined, 374
pure specifier (`= 0`), 333–334

Q

Qt, 29
Qt Creator IDE, 26
question mark (?), 99
queues, 670–672
defined, 670
operations with, 671
specialized, 804–806
quotient, 86

R

\r (carriage return) character, 90–91, 281
rand() function, 223, 301–302
random access iterator, 771–774
random number generators, 776–777
random numbers, 223, 300–302
RandomAccess example, 771–772
RandomNumber example, 301
RandomNumberGenerator example, 776
ranges, 679–680
  defined, 679
  range adapters, 680
  Standard Library, 764–765
Ranges example, 679–680
ranges::size() method, 679–680
raw literals, 784–785
raw pointers
  classes and, 244–248
  creating using initializers, 206–208
  creating using new keyword, 206–208
  defined, 188
  freeing (deleting), 209–211
  problems with, 189–191
  uses for, 188–189
RawAndCooked example, 784
reactive programming, 312
ReadConsoleData example, 278–280
ReadPointer example, 200
ReadString example, 102–103
recursion, 374–375, 379, 423–424
Reddit, 41
Reference01 example, 511
Reference02 example, 512
references, 510–513
  defined, 512
  forward references, 159–161, 237
  nullable references
    optional pointers for, 191
    raw pointers for, 189
  reference variables, 510–511
  relative references, 246
  returning from functions, 511–513
referential transparency, 374
RegEx example, 855–856
regular expressions
  parsing strings with, 850
    adding RegEx library, 851–855
    creating code, 855–857
    pattern matching, 850–851
  Standard Library, 766
reinterpret_cast, 523
relative paths, 846, 853
relative references, 246
remainder (modulus), 78, 86
rename() function, 736
RenameFile example, 735–736
replace() function, 164–165
RepLit IDE, 35
requires element, 401
reset() function, 215–216, 248–249
ret element, 400
Return Value Optimization (RVO), 192
return values
  defined, 150–151
  type of, 152
ReturnDeduction example, 401–402
ReturnPointer example, 221–222
reverse_copy() function, 642–643, 655
reverse-assembly, 783
Rextester.com IDE, 35
right flag, 702–703
rmdir() function, 730
ROOT, 30
runtime libraries, 208
runtime polymorphism, using raw pointers for, 189
rvalue term, 517–519
RVO (Return Value Optimization), 192

S

SayHello project, 55–60
scalars, 497
scientific flag, 702–703
SDL (Simple DirectMedia Layer), 29
seekg() function, 692
semicolon (;), 59, 153, 234
serialization, 738
set class, 658–661
set_intersection() function, 663–664
set_union() function, 664
setf() method, 700–701, 707, 709
sets, 658–661
  intersecting, 662–664
  looking up items, 659–661
  maps vs., 661
  unionizing, 662–664
  unordered, 677–679
    creating, 677
    manipulating, 677–679
Sets example, 658–661
Sets2 example, 662–663
setters, 316–317
  defined, 316
  property vs. method approach, 319
  reasons for using, 317

SetUserName example, 156–157
SetUserName() function, 156–157
setw() method, 708, 710
setx() method, 621–622
SGI (Silicon Graphics, Inc.), 753
shallow copies, 250
shared_ptr pointer, 212, 214–215
SharedPtr example, 214–215
shift-left (SHL) instructions, 478
short keyword, 516
showbase flag, 702–703
showpoint flag, 702–703
showpos flag, 702–703
signed keyword, 516
Silicon Graphics, Inc. (SGI), 753
Simple DirectMedia Layer (SDL), 29
SimpleCast example, 520
SimpleNamespace example, 534–535
SimpleVariable example, 71–73
single quotes (' '), 88, 90
Singleton example, 338
singleton pattern, 337–340
  creating singleton pattern classes, 338–340
  existing patterns, 337–338
sizeof() function, 304
sizeof operator, 484
skipws flag, 702
slicing problem, 566
smart pointers, 191, 211–217
  classes and, 248–249
  creating, 212–216
  defining nullable values using, 216–217
SmartPtr example, 248
sort() function, 413, 668
SortList example, 411–412
source code files, 19–20, 169–186
  .cbp and .cpp files, 20
  creating multiple, 170–179
    adding new files, 170–173
    creating projects with multiple existing files, 173–177
    interaction of multiple files, 177–179
    removing existing files, 173
  downloading, 19
  header wrappers, 185–186
  opening projects, 19–20
  sharing variables among, 183–185
  sharing with header files, 179–183
source files, 240
  defined, 241
  method code, 230
SourceForge, 11, 13, 41
spaceship (<=>) operator, 109–110, 216
specifiers element, 400
SQL (Structured Query Language), 370
square brackets ([ ]), 92, 388
SquirrelClass example, 263–264
srand() function, 301
StackAndQueue example, 671–672
StackOverflow, 42
stacks, 192–194, 196, 469–478, 670–672
  application stack, 470
  defined, 192–193, 470, 670
  operations with, 671
  overflowing, 306
  popping data off of, 196, 470
  pushing data onto, 194, 470
  registers, 470
  stack frames, 193–194, 473, 476–477
  stack pointer, 470
  storing local variables, 473–475
  tracing through assembly code, 475–478
  viewing threads, 475
Standard Library, 3, 374–375, 386–387, 637–680, 753–778
  architecture of, 638
  Boost vs., 818–819
  categories, 755–768
    algorithms, 755–757
    atomic operations, 757–759
    C compatibility headers, 759
    concepts, 759–760
    containers, 760
    coroutines, 760
    <filesystem> header, 761
    input/output, 761
    iterators, 761–762
    localization, 763
    numerics, 763–764
    ranges, 764–765
    regular expressions, 766
    strings, 766
    thread support, 767
    utilities, 767–768
  container classes, 638–674
    comparing instances, 649–654
    copies, 648–649
    copying, 673–674
    defined, 638
    double-ended queues, 669–670
    fixed-size arrays, 642–643
    intersections, 662–664
    iterators, 655–658
    lists, 664–668
    mapping data, 643–644
    Pair template class, 658
    pointers, 644–648
    queues, 670–672
    references, 644–648

Standard Library *(continued)*
stacks, 670–672
storing in sets, 658–661
storing instances, 644–648
unions, 662–664
vectors, 639–642
documentation, 754
dynamic arrays, 675–677
features backported from Boost, 818–819
`find()` algorithms, 774–776
hashes, 768–771
origin of, 753
random access iterator, 771–774
random number generators, 776–777
ranges, 679–680
Standard Template Library vs., 637
temporary buffers, 777–778
unordered sets, 677–679
creating, 677
manipulating, 677–679
User-Defined Literals, 785–791
`std::basic_string` class, 785–787
`std::chrono::duration` class, 789–791
`std::complex` class, 788–789
Standard Library Extensions, 755
Standard Template Library (STL), 637, 753
star (`*`). See asterisk
Start Here window, in Code::Blocks IDE, 21–22
Create a New Project option, 21
Open an Existing Project option, 21
Recent Projects option, 22
Report a Bug or Request a New Feature option, 22
Tip of the Day option, 22
Visit the Code::Blocks forums option, 22
state, 370–371, 381–382
avoiding direct use of, 381–382
defined, 381
effects of lack of, 381
static casts (`static_cast`), 527
static libraries, 48, 809, 813–814
static methods, 506
defined, 509
pointing to, 509–510
singleton pattern, 337–338
`static_cast` (static casts), 527
`StaticCast` example, 528
`StaticMembers` example, 611–612
`std` namespace, 179, 539
`std::array` class, 482, 642–643
`StdArray` example, 642–643
`std::basic_string` class, 785–787
`std::chrono::duration` class, 789–791
`std::complex` class, 788–789
STL (Standard Template Library), 637, 753
streams, 102, 273, 683–726
closing files, 685
containers, 692–693
defined, 683
end of file (EOF) condition, 715–720
EOF check approach, 718–720
record count approach, 715–718
error handling, 693–695
header files, 686–687
input, 711–726
EOF condition, 720–721
extraction operator, 712–714
types, 721–726
`ios` flags, 695–696
making streamable classes, 737–749
manipulating, 742–749
need for, 684–685
opening files, 687–690
output, 697–710
creating fields, 707–710
flags, 700–703
insertion operator, 698–699
setting width, 700, 707–710
specifying precisions, 700, 704–707
paths and filenames, 690
reading from files, 690–692
extraction operator, 712–714
issues with, 720–721
strings, 721–724
structured data, 724–726
text formatting, 738–741
writing to files
insertion operator, 698–699
placing data in specific folders, 699
randomly, 685
sequentially, 685
strings, 721–724
structured data, 724–726
without closing first, 691–692
`string` class, 237
string functions, 163–165
inserting strings into, 163
removing parts of strings, 164
replacing parts of strings, 164
using multiple together, 164–165
`stringify()` function, 807–809
`StringifyInt` example, 807–808
`StringPointer` example, 201
`StringPointer2` example, 202
`StringProcess` example, 437–439
strings, 93–98

accessing individual characters within, 94–95
adding characters to, 96–97
adding strings, 97–98
breaking into tokens, 857–858
changing characters within, 95
defined, 58, 70, 93
parsing with regular expressions, 850
adding RegEx library, 851–855
creating code, 855–857
pattern matching, 850–851
processing to reduce bugs, 437–440
Standard Library, 766
type conversion, 272–276
stringstream string, 273
Stroustrup, Bjarne, 82, 530, 754
Structured Query Language (SQL), 370
structures, 515–539
in C vs. C++, 529–530
casting data, 520–521
const_cast, 528–529
converting data vs., 521–522
dynamically, 524–527
narrowing casts, 523
statically, 527
character types, 516
classes vs., 530
as component data types, 531
equating, 531–532
immutability in, 377–378
lambda expressions and, 407–408
left/right side of assignment statement, 518–520
returning compound data types, 532–534
testing maximum type values, 517–518
StructureTemplate example, 801–802
StructureTemplate2 example, 803–804
subtraction symbol (–), 60
SubtractVariable example, 82–84
suffixes
std::chrono::duration class, 791
std::complex class, 789
User-Defined Literals, 781–784
sum() function, 603, 607–608
swap() function, 214, 216
switch statements, 295–299
switches, defined, 166–167
SwitchStatement example, 296–297
system() function, 734

T

tab (\t) character, 90
tab spaces, 66
template keyword, 607–608
TemplateFromClass example, 626–627
TemplateFromTemplate example, 628–629
templates, 601–636, 795–815
class templates, 798, 804–807
creating templates, 605–607
defined, 601
deriving templates, 623–630
deriving classes from templates, 623–625
deriving templates from classes, 626–627
deriving templates from templates, 627–630
elements of, 797–799
function templates, 630–636, 798–801
overloading and, 632–635
templatizing methods, 635–636
including static members in templates, 611–612
math templates, 799–801
need for, 602–605
parametering templates, 612–622
including multiple parameters, 616–619
non-type parameters, 619–622
putting different types in parameter, 613–615
separating templates from function code, 609–611
specialized, 807–809
structure templates, 798, 801–804
template instantiation, 623
template keyword, 607–608
template libraries, 809–815
coding, 813–814
configuring, 812–813
defining, 810–812
using, 815
typedefs for templates, 622–623
types, 602
when to create, 796–797
temporary buffers, 777–778
TemporaryBuffer example, 777–778
Terminal window, 167
terminate() function, 565
textual input, 435–436
this operator, 417–418
this pointer, 252–256
changes to in C++ 20, 255–256
defined, 253
defining usage of, 253–255
threads
defined, 475
Standard Library support for, 767
viewing, 475

throw() function, 413–414
throw statements, 566–568
ticks, defined, 790
tilde (~), 92
time() function, 301
TIOBE Index, 45
tokenizers, 857–858
tokens, 94
Tokens example, 857–858
toString() method, 239
tparams element, 400
Transform example, 393
transforms, 375, 390
  performing using lambda expressions, 394–395
  using on data points, 393–394
Tronic, 820
try...catch blocks, 558, 563–565
TutorialsPoint IDE, 35
TypeConvert example, 273–275
typedefs, 375, 391–392, 437, 579
  avoiding, 501–502
  templates, 622–623
  using, 499–500
  using multiple, 502–503
typeid() function, 389, 809
typename keyword, 616

U

Ubuntu Software Center, 15–17
UDLs. See User-Defined Literals
#undef directive, 294
Unified Modeling Language (UML), 324
unique_ptr pointer, 212–214
UniquePtr example, 213
unitbuf flag, 702
Unity, 29
unordered sets, 677–679
  creating, 677
  manipulating, 677–679
UnorderedSet example, 677–678
Unreal, 29
unsetf() method, 700
unsigned keyword, 516
uppercase flag, 703
use_count() function, 215
UseAuto example, 404–405
UseNew example, 207
User-Defined Literals (UDLs), 779–794
  cooked literals, 784–785
  developing, 791–794
  need for, 780–781
  prefixes and suffixes, 781–784
  raw literals, 784–785
  in Standard Library, 785–791
    std::basic_string class, 785–787
    std::chrono::duration class, 789–791
    std::complex class, 788–789
UserName example, 154
Username() function, 154
using namespace line, 535–537
UsingAuto example, 145

V

values
  assignment (setting), 72–73
  changing, 74
  defined, 71
  retrieving, 73
VariableAddress example, 197
VariablePointer example, 218
VariablePointer2 example, 219–220
variables, 69–91
  aspects of, 70–71
  assignment (setting), 72–73
  Boolean, 100–101
  changing values, 74
  characters, 88–91
    carriage return character, 90
    defined, 88
    initializing, 88
    newline character, 89–90
    nonprintable characters, 89
    null character, 89
    tab character, 90
    values, 88
  declaring multiple, 73
  defined, 69–70
  floating-point numbers, 70
  global, 183–185
  initializing, 75–76
  integers
    adding, 78–82
    creating, 70–73
    defined, 70
    dividing, 86–88
    multiplying, 84, 86
    subtracting, 82–84
  naming, 71–72, 76–77
    bad names, 77
    case sensitivity, 77
    length, 77
    using My in names, 76
  pointing to methods, 506–509
  putting in namespaces, 537
  setting one variable equal to another (copying), 74–75
  sharing among source files, 183–185
  specifying type, 71–72
variadic templates
  lambda expressions, 424–425
  recursion, 423–424
  sending arguments using, 422–425
VariadicTemplate example, 422
VariadicTemplate2 example, 423

VariadicTemplate3 example, 424–425
VarTypes example, 516–517
vector class, 437, 639–642
  deque class vs., 669–670
  list class vs., 669
vectors
  functional programming, 386, 391–392
  storing in, 639–642
Vectors example, 639
Vectors2 example, 640–641
Vectors3 example, 648–649
VehicleClass example, 265–266
views::filter() method, 680
virtual inheritance, 584–588
virtual keyword, 330–331, 584
virtual methods, 330–334
Visual C++ debugger, 455
Visual Studio Code, 28
Vlissides, John, 336
volatile keyword, 528

W

WalnutClass example, 260–261
Wandbox, 415, 679, 729
while loops, 130–132, 655, 724
  conditions in, 130–131
  defined, 116–117
  done variable, 131–132
  when to use, 117
WhileLoop example, 130–131
WhileLoop2 example, 132
width() method, 707
WidthFunction example, 708–709
Windows, installing Code::Blocks IDE, 12–13
Windows Calculator, 195–196
Wine, 29
WriteReadString example, 721–723
WriteReadStucture example, 724–726
wxWidgets plug-in, 23

X

XML, 841
XSL (DocBook eXtensible Stylesheet Language), 841