Symbols and Numerics

- (minus sign), 40, 261
-- operator, 61
! (exclamation point), 66, 626
!= (not equal) operator, 79
# (pound sign), 66, 69, 261, 793
$ (dollar sign), 792
% (percent sign), 62
& (ampersand)
  getting variable address, 150
  passing by reference, 171
  pointers to functions, 445
  symbol names, 66
  template functions, 537–538
  using in Memory window, 436
&& (and operator), 81
() parentheses
  casting types, 218
  comparison expressions, 72
if statements, 82
in Makefiles, 792
order of mathematics operations, 44
* (asterisk)
  comments, 213
  declaring pointers, 150
  mathematics operations, 40, 61
  UML drawings, 303, 304
*= operator, 61
. (dot), 66, 122, 184, 491
... (ellipsis), 502
/ (forward slash)
  comments, 212–213
  mathematics operations, 40, 62
  pathnames, 622, 660
: (colon)
  accessing member functions, 188
  assignment statements, 324
  conditional operations, 72
  fully qualifying names, 523
  template functions, 537
; (semicolon)
  class definitions, 181
  function prototypes, 119
  functions and, 113
  loop statements, 86
  statement syntax, 37, 40
? (question mark), 72
@ (at symbol), 66
[ ] square brackets
  accessing characters in strings, 67
  declaring arrays, 424
  in lambda expressions, 602
  symbol names, 66
  using with map type, 513
\ (backslash)
  escape character, 45
  in Makefiles, 790, 793
  pathnames, 622, 660
  tab character, 65
  (caret), 66
  (underscore), 55
{ } curly braces
  in code editor, 34
  initializing arrays, 430
  in lambda expressions, 602
  symbol names, 66
  in UML diagrams, 335, 338
| | (or operator), 80
~ (tilde), 66
+ (plus sign), 40, 261
++ operator, 58–59, 92
+= operator, 58–59, 69–70
< (less-than symbol), 37, 79, 575
< > angle brackets
  including header files, 140–141
  using with templates, 535, 537, 540, 555
<< (insertion operator)
  defined, 75
  manipulators and, 673–675
  overloading in class, 670–672
  using with stream, 631–632
<< >> (stereotyping symbol), 308
address-of operator, 445
adjacent_difference( ) function, 686
adjacent_find( ) function, 686
Adobe Acrobat, 738
advance( ) function, 686
aggregation, 305–307
algorithm directory, Boost, 741
algorithms, Standard Library, 686–687
aliases
object, 209–210
reference as, 450
alignment flags, 637
all rule, Makefile, 793
allocate( ) function, 691
allocation of memory
defined, 146
dynamic, 157
using initializer, 159–160
using new operator, 157–159
allocators
polymorphic, 691
Standard Library, 690–691
American National Standards Institute.
See ANSI
ampersand (&)
getting variable address, 150
passing by reference, 171
pointers to functions, 445
symbol names, 66
template functions, 537–538
using in Memory window, 436
analysis workflow, 275–277
and operator (&&), 81
angle brackets (< >)
including header files, 140–141
using with templates, 535, 537, 540, 555
anonymous functions, 599
ANSI (American National Standards
Institute)
adjusting access to class members, 508
GCC and, 455
initializers, 363–364
obtaining documentation, 684
Standard Library, 561
standardization of C++, 1
virtual keyword, 519
app flag, 627–629
appending information, 621
-apple_macro switch, 755
applications
adding library to project, 767–770
breaking, 399
building
errors during, 37–38
overview, 33–34
using Boost library, 762–764
bulletproof, 385
compiling
conditional compilation, 225–226
defined, 46
explained, 784
including debugging information, 389
keyboard shortcuts, 46
make utility and, 785–786
console, 29
Debug version location, 32
documenting, 371
feasibility, 294
flow control
calling functions, 106–107
dividing work using functions, 101–106
do-while loop, 94–95
else if statements, 83–84
else statement, 82–83
evaluating conditions, 78–79
evaluating multiple conditions, 80–81
if statements, 81–82
for loop, 86–92
loops, 95–97
real-world comparison, 78
while loop, 92–94
graphical, 22
lifecycle, 287
mission-critical, 125
planning classes for, 272–275
Release version location, 32
resources for, 785
running
in Code::Blocks IDE, 33–34, 39
keyboard shortcuts, 46
process overview, 46
state, 287–288
types of, 28
workflow for building, 274–275
arguments
command-line, 398
function
command-line, 125–127
lambda expressions, 602
parameters versus, 112
argv variable, 437
ARM (Acorn RISC Machine) processors, 10
array directory, Boost, 741
arrays
accessing elements, 240–241
adding and subtracting pointers, 244–246
allocating memory on heap, 159, 433–435
of arrays, 432, 435–437
best practices, 428
bounds, 431
changing array pointed to, 443
calling IDE for, 594
dynamic
configuring IDE for, 594
declaring, 595–596
heap and, 594
overview, 594
elements in, 240
index number for, 240, 385–386, 424
multidimensional
initializing, 430
overview, 429–430
passing, 430–432
name for, 425, 427
passing to function, 243–244
of pointers, 435–437
pointers and, 425–428
size of, 424, 429–430
storing pointers in, 241–243
using BOOST_FOREACH macro with, 778
vector type versus, 563
arrow keys, 35
\-ascii switch, 755
Assembler language, 784
assembly
  defined, 414
  tracing code, 417–420
\-assert_macro switch, 755
assignment, variable, 50, 80, 458–459
association between classes, 302
asterisk (\*), comments, 213
declaring pointers, 150
mathematics operations, 40, 61
in UML drawings, 303, 304
at symbol (@), 66
ate flag, 629
atomic activities, 326
attrib member, 664
attributes
  class members and, 304
  composition and, 306
  object, 249
auto keyword
detecting variable type created
  by UDL, 706
lambda expressions and, 605–606
uses for, 607
AutoIndex tool, 750

\B
B2 command, 753
back_insert_iterator type, 686, 690
backslash (\)
  as escape character, 45
  in Makefiles, 790, 793
  pathnames, 622, 660
tab character, 65
bang operator (!), 66, 626
bars in diagram, 320
base class
  defined, 208
designing, 260–261
diagrams displaying, 282
initializing members first, 483
multiple inheritance, 514–516
virtual inheritance, 516–520
bases, formatting, 637
basic_filebuf class, 629
basic_ostream class, 632
basic_string class, 685, 708–710
batch files, 124
bcp tool, 750, 759–761
begin ( ) function, 574–576, 778
bidirectional association, 314
bidirectional_iterator_tag type, 686

\b
binary_compose type, 688, 690
binary_function type, 688
binary_negate type, 688, 690
binary_search ( ) function, 686
binder1st type, 688, 690
binder2nd type, 688, 690
bit_vector type, 685
bitset type, 685
black box, classes as, 257
Blag, 14
boilerplate code, 776
Booch, Grady, 282
bool type, 73–74, 456
boolalpha flag, 635, 637
Boole, George, 74
Boolean variables
  conditional operators and, 73
  overview, 74, 456
Boost library. See also Standard Library
  applications using, 738
AutolIndex tool, 750
bcp tool, 750, 759–761
BOOST_FOREACH macro, 776–778
BOOST_REVERSE_FOREACH macro, 777–778
BoostBook tool, 750, 757–758
Build tool
  defined, 749
  examples for, 753–754
  purpose of, 752
  using, 753
building application using, 762–764
building libraries, 743–744
building tools, 749–752
categories listing, 765
classes included in, 765
compiler support, 739
configuring Code::Blocks IDE, 744–749
clean section, Makefile, 792
class, 773–775
directory structure, 741–742
documentation, 740
downloading, 738, 740
examples included, 765
features, 738–739
Filesystem library, 778–781
header-only libraries, 742
including header files, 764
Inspect tool, 750, 755–756
installing, 744
Jam tool, 749
lack of support on some Linux distributions, 14
licensing, 739–740
namespaces from, 764
numeric conversion, 773–776
overview, 738
paid support, 738, 740
previous versions, 743
QuickBook tool, 750, 758–759
RegEx library
adding library to project, 767–770
defined, 743
documentation for, 766
using, 770–772
Regression tool, 750, 754
Release tool, 750
Standard Library and, 779
System library, 778
testing installation, 744–749
tokenizer class, 772–773
unpacking, 741–742
using classes from, 764
Visual Studio support, 740
Wave tool, 750, 761
--boost switch, 760
BOOST_FOREACH loop, 776–777
BOOST_REVERSE_FOREACH macro, 777–778
BoostBook tool, 750, 757–758
BoostPro Computing, 740
Borland C++ Builder
Boost support, 739
call stack, 414
clean section, Makefile, 792
dlvalue and rvalue expressions, 458
Standard Library and, 561
std namespace and, 475
Borland Delphi, 414
bounds, array, 431
break statement
checking for EOF, 653
nested loops and, 98–99
switch statements, 234
using, 95–96
breaking applications, 399
breakpoints
changing variable values, 408–409
defined, 400
disabling, 402–404
enabling, 402–404
overview, 390, 399
setting in Code::Blocks, 401–402
variables and, 404
watching objects, 407–408
watching variables, 406–407
window in Code::Blocks, 396
-brief switch, 756
bugs. See also debugging
anticipating
case sensitivity, 380–381
data input errors, 378–379
processing strings, 382–384
typos, 380–381
while writing code, 385–386
working with arrays, 428
bulletproof application and, 385
features versus, 375–377
history of, 375
Build tool
defined, 749
examples for, 753–754
purpose of, 752
using, 753
building applications
  in Code::Blocks IDE, 33–34
  errors during, 37–38
  using Boost library, 762–764
building Boost libraries, 743–744
building Boost tools, 749–752
business process reengineering, 272–275

c
C++
  advantages of, 1
  comments, 211–213
  concise code, 600
  history of name, 59
  numbering positions in string, 68
  obtaining, 10
  size of, 423
  version 11 features
    dynamic arrays, 594
    lambda expressions, 599
    polymorphic allocators, 691
    UDLs, 703
    unordered_set type, 597
  version 14 support, 709
  version used in book, 9
C++ Development Tooling (CDT), 25
call stack. See stack
Call Stack window, Code::Blocks IDE, 396, 412–414
capabilities, object, 249, 252–253
capture clause, lambda expression, 602, 609–610
car model analogy, 356–360
caret (^), 66
caret (cursor), 34
carriage return character, 65
CASE (Computer-Aided Software Engineering), 284, 334, 338
case keyword, 234
case sensitivity, 54, 380–381
casting, type
  defined, 217
dynamic cast keyword, 461–465
numeric conversions, 218
overview, 460–461
static_cast keyword, 465–466
catch block
  catching any exception, 502–503
  catching references, 501–502
  mandatory with try block, 465
  rethrowing same exception, 503–504
  using, 499–500
.cbp file extension, 19
CDT (C++ Development Tooling), 25
Ceil rounding, 775
CentOS, 14
central processing unit. See CPU
char type, 64, 455, 708, 778
char_produtor type, 685
char16_t type, 705, 708
char32_t type, 705, 708
characters
  accessing in strings, 67
  BOOST_FOREACH macro, 778
  defined, 64
  eliminating improper, 384
  nonprintable, 64–66
  null, 64
  UDLs using, 708
chart_traits type, 689
child class, 208
Chop() function, 664
Chrono library, 743
chrono::duration class, 712–713
Church, Alonzo, 599
cin object
  defined, 75
  reading from console, 219
class keyword, 521, 544
classes. See also functions; templates
  abstract, 270–272
  access methods, 257
  accessing members, 188–191
  base class
    defined, 208
    designing, 260–261
    diagrams displaying, 282
    initializing members first, 483
  as black box, 257
  Boost library, 764
capabilities explained, 252–253
constructors
calling one from another, 489–490
copy, 490–492
default, 484–486
default parameters, 483–484
functional, 486–489
handling failures, 492–493
implicit default, 485
initializer list, 480
initializing base class values first, 483
initializing members, 479–484
inline, 479
invoking, 160
order, 497
overloading, 478–479
overview, 201–202, 477–478
parameters for, 205–206
private, 346
using, 203–204
creating custom templates, 725–728
declaring types in, 526–528
defined, 187
deriving
class from template, 548–550
class template from class template,
552–554
overriding function in, 511–514
overview, 207–208
protecting members in, 262
template from class, 550–552
designing, 256–259
destructors
ordering, 497
overview, 202, 477–478
using, 203–204, 493–495
virtually inheriting, 495–497
diagram of, 285
documenting, 371
encapsulation and, 254–256
explained, 175–176
files for, 178
friend keyword, 520–521
hierarchies of, 206–207
inheritance
designing base class, 260–261
displaying in UML, 304–305
multiple, 514–516
overriding member functions, 267–269
overview, 207–208
polymorphism, 269–270
protecting class members, 262–267
types of, 208
virtual, 516–520
instances
analog, 250, 253
preventing creation of, 525
temporary, 486–487, 489
mailbox analog, 250–254
member functions
overloading, 199–201
overriding, 267–269, 511–514
pointer to, 446–449
pointer to static, 449
members
accessing, 188–191, 257
activity diagram and, 287
adjusting access to, 508–510
attributes and, 304
designing base class, 261
designing encapsulation, 256–259
destructors and, 494
drawing static members in UML,
339–340
encapsulation and, 255
explained, 178–180
friend classes and, 520–521
initializing, 479–484
initializing in constructor, 479–484
overview, 187–188
private, 189
protected, 208
protecting, 262–267
protecting when inheriting class,
262–267
public, 189
read-only, 255
templatizing function, 559–560
templatizing member functions,
559–560
members, static
copying instances and, 492
drawing in UML, 339–340
pointer to function, 449
classes (continued)
  scope of, 339
  in template, 538–540
modeling
  collection of classes as model, 284
  design patterns, 343–344
  design workflow and, 277
  Façade pattern, 361
  finding missing, 277
  Observer pattern, 348–356
  overview, 275–276
  searching for nouns and verbs, 276
  Singleton pattern, 344–348
  using CRC cards, 276–277
  using mediator class, 356–371
naming, 178, 250
nesting, 521–526
  object aliases, 209–210
  organizing things using, 177
  parameterized, 340–341
  parts of, 187–188
  passing objects to functions, 194–195
  persistent, 669
  planning for application, 272–275
  pointers and, 191–194
  polymorphism, 269–270, 505–508
  public, 180
  runtime and, 530–531
  separating member function code, 185–187
  singleton, 177, 345
  struct type and, 467
  this pointer, 196–199
  using, 180–184
  using namespaces with, 471–473
  wrapping enum type, 236–237
classifiers, 338
  clean section, Makefile, 792
code reuse, 106
Code::Blocks IDE
  adding library to project, 767–770
  alternatives to, 25
  associating C++ files with, 17
  breakpoints
    changing variable values, 408–409
    commands for, 390
  disabling, 402–404
  enabling, 402–404
  setting, 401–402
  variables and, 404
  watching objects, 407–408
  watching variables, 406–407
  building application, 33–34
  Call Stack window, 396, 412–414
  CASE plugin and, 334
  code editor in, 34–35
  command-line parameters in, 126, 433
  compilers, selecting, 23–24
  compilers and, 10
  compiling with debugging information, 389
  CPU Registers window, 396, 419–420
  creating project from multiple source code files, 133–135
  Current Stack Frame window, 396
  Debugger tab in, 391
  debugging tools in, 390, 395
  debugging using command line arguments, 398
  debugging windows in, 396–397
  Disassembly window, 394, 396, 417–419
downloading, 10–11
dynamic array support, 594
Explorer view, 33
  forums for, 21
  FPU Status window, 397
  hash type support, 692
  installing
    on Linux, 14–16
    on Mac, 13
    on Windows, 11–13
  instruction pointer, 390
keyboard shortcuts, 35
library creation
  adding template, 734–735
  creating, 730–732
  setting up, 732–734
  using, 735
Loaded Libraries window, 397
Logs & others window, 22–23
make utility and, 783
  Management window, 21–22
Memory Dump window, 396, 436
modified files in, 42
opening sample projects, 19–20
platform support, 2
projects
adding library to, 767–770
creating, 21, 28–33
creating from multiple source code files, 133–135
opening, 19–20
overview, 28
removing files from, 132
splitting into multiple source code files, 130–132
types of, 29–30
running application, 33–34
Running Threads window, 396, 416
Signal Handling window, 397
Standard Library and, 561
Start Here window, 20–21
starting for first time, 17–19
supporting newer C++ features in, 691
Targets and Files window, 397
template errors, 543–544
testing Boost installation, 744–749
troubleshooting settings on Windows, 18–19
Watches window, 396
Wave tool and, 761
CodeLite, 25
corercion, type, 217
cohesion, 367
collaboration diagrams, 286, 325–326
colon ():
  accessing member functions, 188
  in assignment statements, 324
  conditional operations, 72
  fully qualifying names, 523
  template functions, 537
COM (Component Object Model), 307
comma (,), 430
command-line arguments
  arrays and, 432–433
  for debugging, 398
  overview, 125–127
comments
  in Makefiles, 793
  syntax for, 211–213
compilers
  advantages of C++, 1
  auto keyword, 605
  C++ 14 support, 709
  comments and, 212
  compiling process, 46, 783–785
  conditional compilation, 225–226
  Demangle() function, 706
differences in interpretation, 10
GCC
  adjusting member access, 508
  Boost support, 739
  Build tool, 753
  C++ 14 support, 709
  debugging information, 389
downloading, 10
  lvalue and rvalue expressions, 458
  non-standard suffix extensions, 707
std namespace and, 475, 621
stream libraries, 619
supporting STL features, 689
header files and, 139
IDE versus, 10
including debugging information, 389
keyboard shortcuts, 46
linker and, 784
listing dependencies, 790–791
make utility and, 785–786
purpose of, 37
return type for lambda expressions, 602–604
selecting in Code::Blocks IDE, 23–24
separating function code from
template, 537
templates and, 529
UDLs and, 703
complex class, 707, 711–712
component diagrams, 285, 307–310
Component Object Model (COM), 307
compose( ) function, 689
composition, 302, 303, 305–307
Computer-Aided Software Engineering (CASE), 284, 334, 338
concatenation, 70
concise code, 600–601
Concurrent Versions System (CVS), 760
conditional compilation, 225–226
conditional operators
  Boolean variables and, 73
  numerical conditions, 79–80
overview, 71–73
conditions
  defined, 77
else if statements, 83–84
else statement, 82–83
  evaluating, 78–79
if statements, 81–82
for loop, 86, 87–88
  multiple evaluations, 80–81
switch statements, 233–236
conio library, 219
console
  applications, 29
  cout, 39
  delimiters in, 219
  output to, 37–40
  reading from, 74–76, 219–223
  tab characters in output, 44–45
const keyword
  copying instances and, 492
  creating constants using, 230
  passing by reference, 172–173
  using parameters in functions, 195–196
  using with arrays, 439
const_iterator type, 778
constants
  arrays, 438–439
  overview, 230–233
construct( ) function, 690
construction phase, 295–296
constructors
  calling one from another, 489–490
  copy, 490–492
  default, 484–486
  default parameters, 483–484
  functional, 486–489
  handling failures, 492–493
  implicit default, 485
  initializer list, 480
initializing base class values first, 483
initializing members, 479–484
inline, 479
invoking, 160
ordering, 497
overloading, 478–479
overview, 201–202, 477–478
parameters for, 205–206
private, 346
using, 203–204
container classes
  defined, 511
objects and, 250
Standard Library
  comparing instances, 570–574
  copy constructor for, 592–594
deque type, 589
  equal operator for, 592–594
  instances in, 566–569
  iterating through, 574–577
  library architecture, 562
list type, 584–588
  listing of, 685
  pointers in, 568–569
queue type, 589–592
  references and, 567–568
set type, 578–584
stack type, 589–592
unordered_set type, 596–598
  as template parameter, 544
using BOOST_FOREACH macro with, 778
Context library, 743
continue statement
  nested loops and, 98–99
  using, 96–97
control characters, 708
conversion UDLs, 714–715
conversion_traits object, 775
convert( ) function, 775
converter class, 773–775
converting types
  dynamic cast keyword, 461–465
  numeric conversion, 773–776
  overview, 213–218, 460–461
static_cast keyword, 465–466
cooked literals, 707–708
copy command, 125, 787
copy constructors, 490–492, 592–594
copy( ) function, 686
copy_backward( ) function, 687
copy_n( ) function, 687
CopyFile( ) function, 664
copying variables, 52–53
-copyright switch, 755
count( ) function, 597, 687
count_if( ) function, 687
counter variable, 85, 86, 91–92
counting backward
  using BOOST_REVERSE_FOREACH macro, 777–778
  using for loop, 88–89
coupling, 367
cout object
  adding capabilities to class, 238
  overview, 39
cp command, 125, 665, 787
cpp command, 224
  .cpp file extension, 20, 178
CPU (central processing unit)
  local variable storage, 415
  polling and, 349
  stack and, 411–412
CPU Registers window, Code::Blocks IDE, 396, 419–420
create_directory( ) function, 781
CreateFile( ) function, 625
-crif switch, 755
ctime( ) function, 664
Ctrl key, 35
curly braces ({ })
  in code editor, 34
  initializing arrays, 430
  in lambda expressions, 602
  symbol names, 66
  in UML diagrams, 335, 338
Current Stack Frame window, Code::Blocks IDE, 396
cursor, 34
CVS (Concurrent Versions System), 760
-cvs switch, 756
--cvs switch, 760
Cygwin, 389, 397, 561, 788, 790

D

d suffix, 707
data structures
arrays
  accessing elements, 240–241
  adding and subtracting pointers, 244–246
  allocating memory on heap, 159, 433–435
  of arrays, 432, 435–437
  best practices, 428
  bounds, 431
  changing array pointed to, 443
  command-line parameters and, 432–433
  constant, 438–439
  declared size versus none given, 426
  declaring, 240–241, 424–425
  defined, 126
  dynamic, 594–596
  elements in, 240
  index number for, 240, 385–386, 424
  multidimensional, 429–432
  name for, 425, 427
  passing to function, 243–244
  of pointers, 435–437
  pointers and, 425–428
  size of, 424, 429–430
  storing pointers in, 241–243
  using BOOST_FOREACH macro with, 778
vector type versus, 563

struct type
  creating custom templates, 722–725
  equality between, 469–470
  overview, 466–468
  packaging other data types as, 468–469
  returning from function, 470–471
  templates for, 719

data types
  auto keyword and, 605
  back_insert_iterator type, 686, 690
  bidirectional_iterator type, 686
  bidirectional_iterator_tag type, 686
data types (continued)
  binary_compose type, 688, 690
  binary_function type, 688
  binary_negate type, 688, 690
  binder1st type, 688, 690
  binder2nd type, 688, 690
  bit_vector type, 685
  bitset type, 685
  bool type, 73–74, 456
  casting
    defined, 217
    dynamic_cast keyword, 461–465
    multiple inheritance and, 519
    numeric conversions, 218
    overview, 460–461
    static_cast keyword, 465–466
  char type, 64, 455, 708, 778
  char producer type, 685
  char16_t type, 705, 708
  char32_t type, 705, 708
  chart_traits type, 689
  const_iterator type, 778
  converting, 460
    _Decimal32 type, 707
    _Decimal64 type, 707
    _Decimal128 type, 707
  declaring within classes, 526–528
deque type
  overview, 589
  queue type and, 590
  stack type and, 590
  Standard Library containers, 685
  vector type versus, 590
divides type, 688
double type
  converting to int, 773–775
  creating complex numbers, 712
  creating conversion UDL, 714
  customizing templates for data
types, 729
  defined, 456
  non-standard suffix extensions, 707
  precision for, 637–640
  pseudorandom numbers, 699
  UDL suffixes, 705
dynamic_cast keyword, 461–465
elem type
  declaring inside class, 528
  drawing in UML, 338–339
  naming, 180
  overview, 177
  wrapping with class, 236–237
equal_to type, 688
  float type, 456, 705, 707, 712
    __float80 type, 707
    __float128 type, 707
  forward_iterator type, 686
  forward_iterator_tag type, 686
  front_insert_iterator type, 686, 690
  greater type, 688
  greater_equal type, 688
  hash type, 685, 688, 691–694
  hash_map type, 692
  identity type, 688
  input_iterator type, 686
  input_iterator_tag type, 686
  insert_iterator type, 686, 690
  int type, 49, 455, 704–705, 773–775
  integers
    adding together, 56–59
    casting, 466
    converting double type to, 773–775
    converting to floating-point
    numbers, 216
    converting using Boost library, 741
    dividing, 62–63
    format flags, 634–637
    mathematics using, 55
    max( ) function, 700
    min( ) function, 700
    multiplying, 61
    random number as, 699
    subtracting, 60–61
    UDL suffixes, 704–705
    variables for, 48–50
  istream_iterator type, 686
  istringstream type, 214
  iterator type, 574, 581, 773, 778
  iterator_traits type, 686
  less type, 688
  less_equal type, 688
  list type
deque type versus, 589
insert( ) function, 585
overview, 584
push_back( ) function, 584
push_front( ) function, 584
queue type and, 590
stack type and, 590
Standard Library containers, 685
using, 353–354, 585–587
using iterator with, 585
vector type versus, 588
logical_and type, 688
logical_not type, 688
logical_or type, 688
long double type, 456, 712, 714
long int type, 456, 699, 704–705
map type
begin( ) function, 574–576
comparing instances, 570–574
end( ) function, 575
instances and, 569
iterating through, 574–577
overview, 512–513
Pair class and, 578
pointers and, 569
set type versus, 581
sort order, 571–572
Standard Library containers, 685
using, 565–566
using with template parameters, 544
mem_fun_ref_t type, 688, 690
mem_fun_t type, 688, 690
mem_fun1_ref_t type, 688, 690
mem_fun1_t type, 688, 690
minus type, 688
modulus type, 688
multimap type, 685
multiplies type, 688
multiset type, 685
negate type, 688
not_equal_to type, 688
omanip type, 674–675
ostream_iterator type, 686
ostringstream type, 169, 214, 728
output_iterator type, 686
output_iterator_tag type, 686
packaging as struct type, 468–469
pair type, 689, 701
plus type, 688
pointer_to_binary_function type, 688, 690
pointer_to_unary_function type, 688, 690
priority_queue type, 685, 690
project1st type, 688
project2nd type, 688
queue type, 589–592, 685, 690, 727
random_access_iterator type, 686
random_access_iterator_tag type, 686
ratio type, 712
raw_storage_iterator type, 686, 690, 691
reverse_bidirectional_iterator type, 686, 690
reverse_iterator type, 686, 690
rope type, 685
runtime and, 530–531
set type
find( ) function, 580–581
finding intersection, 583–584
map type versus, 581
overview, 578
Standard Library containers, 685
unionizing, 581–584
using, 579–580
short int type, 455
signed type, 456–457
slist type, 685
stack type, 589–592, 685, 690
static_cast keyword, 465–466
string type, 67
strings
accessing characters in, 67
adding onto, 69–70
basic_string class, 708–710
changing part of, 69
class for, 183
combining, 70–71
data types (continued)
- constant, 70
- converting, 214
- converting using Boost library, 741
- creating tokens from, 772–773
- defined, 39
- delimiters, 68
- functions for, 122–124
- getting part of, 68–69
- index number for, 385–386
- modifying using pointer, 167
- numbering positions in, 68
- overview, 67
- pointing to, 153–155
- processing to anticipate bugs, 382–384
- random, 169
- raw and cooked processing, 707–708
- shortcut notation, 167
- using BOOST_FOREACH macro with, 778
- using new operator with, 160–161
- variables for, 48

struct type
- creating custom templates, 722–725
- equality between, 469–470
- overview, 466–468
- packaging other data types as, 468–469
- returning from function, 470–471
- templates for, 719

unordered_map type, 692
unordered_set type
- C++ 11 extensions, 597
- overview, 596–597
- using, 597–598

unsigned type, 154, 456–457, 704–705

vector type
- array versus, 563
- begin( ) function, 574–576
- defined, 536
- deque type versus, 589, 590
- end( ) function, 575
- iterating through, 574–577
- list type versus, 588
- size of, 577
- stack type and, 590
- Standard Library containers, 685
- using, 562–564
- using random access iterator, 694–696
- void type, 456, 458
- wchar_t type, 456, 705, 708, 778

aa suffix, 707
deallocate( ) function, 691
Debian, 14
--debug switch, 759
debug version
- #define statement, 225–226
- defined, 131
- location for, 32
debugging
- available tools for, 397–398
- breakpoints, 390
- changing variable values, 408–409
- disabling, 402–404
- enabling, 402–404
- setting, 401–402
- variables and, 404
- watching objects, 407–408
- watching variables, 406–407
- command-line arguments for, 398
- compiling for, 389
- debuggers, 387–388
- example of process, 392–395
- history of, 375
- lvalue and rvalue in error messages, 458–459
object files and, 784
process overview, 391–392
starting debugger, 389
stepping into/over, 392
tab in Code::Blocks, 391
template errors, 543–544
tracing assembly code, 417–420
viewing stack, 412–414
viewing threads, 416
windows in Code::Blocks, 396–397
de决 flag, 636
decimal numbers
casting, 466
converting to hexadecimal, 148
converting to integers, 216
creating complex numbers, 712
defined, 48
double type, 456
float type, 456
format flags, 634–637
long double type, 456
specifying precision for output, 637–640
decimal point, 48
Decimal to Hex Converter application, 148
_decimal32 type, 707
_decimal64 type, 707
_decimal128 type, 707
decision in flowcharts, 281
declaring variables, 49–51
decrementing of stack, 416
def_overflow_handler, 775
default constructor, 484–486
default keyword, 234
default parameters
in constructor, 483–484
for template, 575
#define directive, 143–144, 224–226
delete operator
arrays on heap and, 434–435
for each new operator, 386
freeing memory, 161–162
delete[] operator, 434, 596
delimiters, 68, 219, 654
Delphi, 324
Demangle( ) function, 706, 710
demancellors, 637
dependencies, 790
deployment diagrams, 285, 310–311
-deprecated_macro switch, 755
deque type
overview, 589
queue type and, 590
stack type and, 590
Standard Library containers, 685
vector type versus, 590
dererencing pointer, 151
derived classes
class template derived from class, 550–552
class template derived from class template, 552–554
multiple inheritance, 514–516
overriding function in, 511–514
overview, 207–208
protecting members, 262
from template, 548–550
virtual inheritance, 516–520
design patterns
Façade pattern, 361
Observer pattern
automatically adding observer, 354–356
overview, 348–353
Standard Library and, 353–354
overview, 343–344
Singleton pattern, 344–348
using mediator class
code example, 361–371
collaborations between objects, 361
overview, 356–360
Design Patterns: Elements of Reusable
  Object-Oriented Software, 343
designing classes, 272–275
destroy( ) function, 321, 690
destructors
ordering, 497
overview, 202, 477–478
using, 203–204, 493–495
virtually inheriting, 495–497
Dev-C++, 25, 389, 398, 561
df suffix, 707
diagrams, UML
  activity
defined, 281
overview, 286–287
reading, 326–328
collaboration, 286, 325–326
component, 285, 307–310
deployment, 285, 310–311
diamond in, 302, 306, 327
dynamic, 284, 286–287
notating, 334–335
object, 286, 313–315
sequence
comparisons in, 323–324
defined, 286–287
loops in, 322–323
notating, 321
overview, 318–321
state, 328–329
statechart, 286–287
static, 284–285
use case
creating, 315–316
defining on paper, 317–318
event flow example, 317
matching with requirements, 318
overview, 286
diamond inheritance, 520
Digital Mars, 739
direct.h header, 633
directories
creating, 660
creating using Boost library, 781
current working directory, 633
deleting, 661
lack of support for, 659
reading contents of, 661–664
renaming, 659, 666–667
root directory access, 649
Disassembly window, Code::Blocks IDE, 394, 396, 417–419
Disk Operating System (DOS), 124
distance( ) function, 686, 687
distance_type( ) function, 686
divides type, 688
division, 40, 62–63
dl suffix, 707
DLL (Dynamic Link Library)
  component diagrams, 286, 307–310
  creating project for, 29
  executable files and, 785
  linkers and, 784
  templates and, 730
DocBook, 750, 757
Document Type Definition (DTD), 757
documentation
  anticipating bugs, 381
  for Boost library, 740
  BoostBook tool, 757–758
  for classes, 371
  for make utility, 793
  QuickBook tool, 758–759
  for RegEx library, 766
  for Standard Library, 684
dollar sign ($), 792
DOS (Disk Operating System), 124
dot (.), 66, 122, 184, 491
dotted lines in diagram, 308
double quote ("), 39, 45, 65, 140–141
double type
  converting to int, 773–775
  creating complex numbers, 712
  creating conversion UDL, 714
  customizing templates for data types, 729
defined, 456
  non-standard suffix extensions, 707
  precision for, 637–640
  pseudorandom numbers, 699
  UDL suffixes, 705
double-precision floating-point, 456
do-while loop
  break statement, 95–96
  continue statement, 96–97
  nesting, 97–99
  overview, 85–86
  using, 94–95
down-cast, 462
downloading
  Boost library, 738, 740
  GCC, 10
drand48( ) function, 699–700
DTD (Document Type Definition), 757
duration_cast, 712
dynamic allocation, 157
dynamic arrays
  C++ 11 extensions, 594
configuring IDE for, 594
declaring, 595–596
heap and, 594
overview, 594
dynamic_cast keyword, 461–465
dynamic diagrams, 284, 286–287
dynamic library, 307
Dynamic Link Library. See DLL

E

EAX (Extended Accumulator), 416, 418, 419
EBP (Extended Base Pointer), 415, 418
ECB (Emacs Code Browser), 25
Eclipse, 25
elaboration phase, 294–295
elements, array, 240–241
ellipsis (…), 502
#else directive, 227
else if statements, 83–84
else statement, 82–83
Emacs, 25
Emacs Code Browser (ECB), 25
embedded classes, 521–526
emplace( ) function, 596, 598
encapsulation
  designing for classes, 256–259
  overview, 254–256
end( ) function, 575, 778
End key, 35, 41
End of File (EOF), 648–653
-end switch, 755
#endif directive, 143–144, 226–227
endl function, 39, 673
enum type
  declaring inside class, 528
drawing in UML, 338–339
naming, 180
overview, 177
wrapping with class, 236–237
EOF (End of File), 648–653
epilog, 415
eqstr structure, 693
equal( ) function, 608, 687
equal sign (=), 52, 72, 80
equal_range( ) function, 687
equal_to type, 688
equal_equal (==), 66, 72, 79
equality
  comparing instances, 570–574
struct type, 469–470
erand48( ) function, 699
erase( ) function, 122, 596, 598
error handling. See exceptions
escape character, 45
escape-sequence, 46
ESP (extended stack pointer), 412
eXtensible Stylesheet Language (XSL), 750, 757
extensions, C++, 594, 597, 599, 691
extern keyword, 142–143, 428
external linkage, 785
ext/functional header, 689
extraction operator (>>)
defined, 75
overloading in class, 670–672
reading from console, 219
using with stream, 645–648

F
F suffix, 705
F8 key, 389
F9 key, 33, 46
F10 key, 33, 46
fabs() function, 105, 689
Façade pattern, 361
fail() function, 625–626
feasibility of project, 294
features
bugs versus, 375–377
user decisions for, 377–378
Fedora, 14
fi suffix, 707
FIFO (First In First Out), 589
File Associations dialog box, Code::Blocks, 17
file_size() function, 781
filebuf class, 629
filenames versus pathnames, 623
Files tab, Code::Blocks, 22
filesystem
copying files, 664–666
current working directory, 633
directories
creating, 660
creating using Boost library, 781
deleting, 661
lack of support for, 659
reading contents of, 661–664
moving files, 666–667
reading and writing files
extraction operator (>>, 645–648
fields, 642–643
finding EOF, 648–653
format flags, 634–637
formatting output, 633–634
handling errors when opening, 624–626
insertion operator (<<), 631–632
ios flags, 627–629
opening file, 621–624
precision for numbers, 637–640
reading formatted input, 657
reading lines, 653–656
setting width, 640–643
renaming files/directories, 659, 666–667
using Boost library classes, 778–781
Files System library, 743, 778–781
fill() function, 687
fill_n() function, 687
FILO (First In Last Out), 589
final state, 327
finalizer, for loop, 86
find() function, 580–581, 597, 598, 687, 697–698
find_end() function, 687, 697–698
find_first_of() function, 687, 697–698
_findclose() function, 662
_finddata_t structure, 662
_findfirst() function, 662
_findnext() function, 662
First In First Out (FIFO), 589
First In Last Out (FILO), 589
first variable, 578
fixed flag, 635, 637
float type, 456, 705, 707, 712
__float80 type, 707
__float128 type, 707
floating-point numbers
casting, 466
converting to integers, 216
creating complex numbers, 712
defined, 48
double type, 456
float type, 456
format flags, 634–637
long double type, 456
specifying precision for output, 637–640
Floating-Point Unit (FPU), 397
Floor rounding, 775
flow, application
  calling functions, 106–107
  dividing work using functions, 101–106
do-while loop, 94–95
do-while loop, 94–95
do-while loop, 94–95
do-while loop, 94–95
else if statements, 83–84
else statement, 82–83
evaluating conditions, 78–79
evaluating multiple conditions, 80–81
if statements, 81–82
for loop
  condition for, 87–88
  counting backward, 88–89
  incrementing one step, 89–90
  multiple counter variables, 91–92
  using, 86–87
loops
  break statement, 95–96
  continue statement, 96–97
real-world comparison, 78
while loop, 92–94
flowcharts, 280–281
flushing files, 619
for loop
  break statement, 95–96
  condition for, 87–88
  continue statement, 96–97
  counting backward, 88–89
  incrementing one step, 89–90
  multiple counter variables, 91–92
  nesting, 97–99
  overview, 85–86
  using, 86–87
  using auto keyword, 607
  using with arrays, 241
for_each( ) function
  BOOST_FOREACH macro and, 776
  lambda expressions and, 601–602
  Standard Library algorithms, 687
format flags, 634–637
forward reference
  defined, 120
  nested classes, 524
  separating class member functions, 185–187
forward slash (/)
  comments, 212–213
  mathematics operations, 40, 62
  pathnames, 622, 660
forward_iterator type, 686
forward_iterator_tag type, 686
FPU (Floating-Point Unit), 397
freeing memory, 161–163
friend keyword, 520–521
front( ) function, 589, 591
front_insert_iterator type, 686, 690
fstream library, 619, 629, 646
fully qualifying names, 472, 523
func keyword, 447–448
function templates
  overloading, 556–559
  overview, 554–556
functional constructors, 486–489
functions
  abstract virtual, 271, 305
  activity diagram and, 287
  address of, 673
  assigning variable to, 111
  calling, 106–107
  color-coding on SGI website, 685
  const parameters in, 195–196
  creating, 109–110, 113
  data input to, 104
  debugging where called from, 414
  defined, 101, 103
  dividing work using, 101–103
  encapsulation and, 256
  forward reference for, 120
  friend keyword, 520–521
  hiding in class, 513–514
  initialization, 202
  local variables, 117–118
  main( ) function, 36, 124–127
  member, 122, 179, 188
  memory address for, 146
  methods and, 304
  multiple source code files and, 130, 136–137
  overloading, 120–121, 199–201
  overriding, 267–269
overview, 104–106
parameters for, 109, 111, 113–116
passing to
arrays, 243–244
multiple variables, 108–109
objects, 194–195
pointers, 163–166
variables, 107–108
pointers to, 445–449
positioning in code, 110
private, 190–191
prototypes, 118–120, 185–187
using, 118–120
using header files, 138
pure abstract, 350
pure virtual, 271
returning from
overview, 104, 109, 111, 112
pointer, 168–170
pointer as nonpointer, 170–171
reference, 451–453
struct type, 470–471
void return, 116–117
reusing, 112
separating class member functions, 185–187
separating from class template, 536–538
stack and, 412
Standard Library algorithms, 686–687
stepping into/over, 392
for strings, 122–124
templates for, 559–560, 719
using namespaces with, 472–473
virtual
abstract, 271, 305
dynamic cast keyword and, 463
polymorphism, 270
pure, 271
functors
defined, 601
Standard Library, 687–689

G

-g option, 389
Gamma, Erich, 343
Gang of Four, 343
GCC (Gnu Compiler Collection)
adjuring member access, 508
Boost support, 739
Build tool, 753
C++ 14 support, 709
debugging information, 389
Demangle( ) function, 706
downloading, 10
lvalue and rvalue expressions, 458
non-standard suffix extensions, 707
selecting in Code::Blocks, 23–24
std namespace and, 475, 621
stream libraries, 619
supporting STL features, 689

gdb, 397
general protection faults (GPFs), 295
generalized class, 302
generate( ) function, 687
generate_n( ) function, 687
Gentoo, 14
get temporary_buffer( ) function, 690, 701
getch( ) function, 221–222
_getch( ) function, 222
getcwd( ) function, 633
getline( ) function, 655–656
GIMP (GNU Image Manipulation Program), 30
global variables, 142–143
GNU (Gnu’s Not Unix), 30
Gnu Compiler Collection. See GCC
GNU General Public License (GPL), 740
GNU Image Manipulation Program (GIMP), 30
__gnu_cxx namespace, 689
Gnu’s Not Unix (GNU), 30
Gödel, Escher, Bach: An Eternal Golden Braid, 288
GPFs (general protection faults), 295
GPL (GNU General Public License), 740
GraphParallel library, 743
greater type, 688
greater_equal type, 688
greater-than symbol (>), 79
gregorian namespace, 764
GTK+ projects, 29, 30
GUI applications, 22
guillemots, 336

H

.h file extension, 138, 620, 694
h suffix, 713
hash type, 685, 688, 691–694
hash_map type, 692
header files
adding only once, 140
brackets versus quotes, 140–141
class definitions, 188
global variables, 142–143
header wrappers, 143–144
header-only libraries in Boost, 742
overview, 138–140
using from Boost library, 764
header line for functions, 113
heap
allocating array on, 433–435
allocating memory on, 159
dynamic arrays and, 594
stack versus, 146–148
Helm, Richard, 343
--help switch, 759
hex flag, 636
Hex to Decimal Converter application, 148
hexadecimal numbers
converting to decimal, 148
overview, 147
hibernation, 376
hierarchies of classes
abstract classes, 270–272
creating inherited class, 207–208
designing base class, 260–261
inheritance types, 208

I

i suffix, 707, 712
IDE (Integrated Development Environment)
alternatives, 25
Code::Blocks as, 9
compiler versus, 10
identifiers, 331
identity type, 688
if statements
defined, 77, 78
else if statements, 83–84
else statement, 82–83
multiple conditions satisfied, 84
overview, 81–82
if suffix, 712
#ifndef directive, 143–144
ifstream class, 620, 624, 651, 654, 655
il suffix, 712
implementation workflow, 275, 296
implicit default constructor, 485
implicit rules, Makefile, 793
in flag, 628–629
inception phase, RUP, 293–294
#include directive
brackets versus quotes, 140–141
functions in separate source code files, 137
header files, 138
overview, 69
purpose of, 223–224
include files. See header files
includes( ) function, 687
incomplete type, 456
increments
  loops, 85, 89–90
  stack, 416
--indent switch, 759
indentation, 41
index, array, 240, 385–386, 424
inequality operator, 80
inference rules, Makefile
  multiple dependencies, 789–791
  overview, 787–788
  rules checked before running, 788–789
inheritance, class
  abstract classes, 270–272
  base class
    defined, 208
    designing, 260–261
    diagrams displaying, 282
    initializing members first, 483
derived class
  class template derived from class, 550–552
  class template derived from class
    template, 552–554
  overriding function in, 511–514
  overview, 207–208
  protecting members in, 262
    from template, 548–550
displaying in UML, 304–305
multiple, 514–516
overriding member functions, 267–269
overview, 207–208
polymorphism, 269–270
protecting class members, 262–267
types of, 208
  virtual, 516–520
initialization function, 202
initializer
  allocating memory using, 159–160
  constructor and, 364
  for loop, 86
initializer list, 480
initializing
  members using constructor, 479–484
  multidimensional arrays, 430
  variables, 53
inline constructor, 479
inline keyword, 729
inner array, 436
inner class, 522–524
inner_product( ) function, 687
inplace_merge( ) function, 687
input
  extraction operator (>>), 645–648
  finding EOF, 648–653
  getting directory contents, 661–664
  reading formatted, 657
  reading lines, 653–656
  validating, 766
input_iterator type, 686
input_iterator_tag type, 686
--input-file switch, 759
insert( ) function, 122, 585, 596, 598
insert_iterator type, 686, 690
insertion operator (<<)
  defined, 75
  manipulators and, 673–675
  overloading in class, 670–672
  using with stream, 631–632
insertion point, 34
Insight debugger, 397, 414
Inspect tool, 750, 755–756
installing
  Boost library
    overview, 743–744
    testing installation, 744–749
Code::Blocks IDE
  on Linux, 14–16
  on Mac OS X, 13
  on Windows, 11–13
Xcode, 13
instance, class
  analogy, 253
  comparing, 570–574
  copying using copy constructor, 490–492
destructors and, 495
overview, 181–182
polymorphism and, 505–508
preventing creation of, 525
runtime and, 530–531
scope for, 339
template instances, 548
temporary, 486–487, 489
throwing directly, 500–501
using in Standard Library container classes, 566–569
instantiating templates, 547
instruction pointer, 390
instructions, 36
int type, 49, 455, 704–705, 773–775
integers
adding together, 56–59
casting, 466
converting double type to, 773–775
converting to floating-point numbers, 216
converting using Boost library, 741
dividing, 62–63
format flags, 634–637
long int type, 456
mathematics using, 55
max( ) function, 700
min( ) function, 700
multiplying, 61
random number as, 699
short int type, 455
subtracting, 60–61
UDL suffixes, 704–705
variables for, 48–50
Integrated Development Environment.
   See IDE
types
   designing encapsulation for class, 256–259
documenting work, 371
encapsulation and, 256
to set of classes, 521
International Standards Organization (ISO), 1, 222, 683, 684
intersection, finding for set types, 583–584
invoking constructor, 160
io.h header, 659
iomanip library, 640
ios class, 623, 634
ios_base class, 627, 634
iostream library, 619, 646
IOStreams library, 743
iota( ) function, 687
is_equal( ) function, 691
is_heap( ) function, 687
is_sorted( ) function, 687
ISO (International Standards Organization), 1, 222, 683, 684
istream class, 651, 655
istream_iterator type, 686
istringstream type, 214
iter_swap( ) function, 687
iterations. See also loops
   through Standard Library container templates, 574–577
   using list type, 585
   using UML, 290–292
iterator type, 574, 581, 773, 778
iterator_category( ) function, 686
iterator_traits type, 686
iterators
   random access iterator, 694–696
   Standard Library, 685–686

J

Jacobson, Ivar, 282
Jam tool, 749
Johnson, Ralph, 343
jrand48( ) function, 699

K

keyboard shortcuts, 35, 46
key/value pairs, 512–513, 544, 565–566, 578

L

L prefix, 705
L suffix, 704, 705
lambda expressions
   auto keyword, 605–606
   C++11 extensions, 599
capture clause for, 609–610
concise code, 600–601
history of, 599
multiple inputs for, 607–609
parts of expression, 601–602
lambda expressions (continued)
  return type
    specifying, 604–605
    using compiler detection, 602–604
    sorting data using, 610–612
    throwing exceptions from, 612–613
lcong48( ) function, 700
left flag, 636, 641, 642
length( ) function, 154
less type, 688
less_equal type, 688
less-than symbol (<), 37, 79, 575
lexicographical_compare( )
  function, 687
lexicographical_compare_3way( )
  function, 687
lib directory, Boost, 741
.lib file extension, 784
libboost_filesystem-mgw47-mt-1_55.a library, 778
libboost_regex-mgw47-mt-1_55.a library, 768
libboost_system-mgw47-mt-1_55.a library, 778
libraries. See also Boost library; Standard Library
  component diagrams, 307–310
  creating for template
    adding template, 734–735
    creating project, 730–732
    overview, 730
    setting up project, 732–734
    using library, 735
defined, 28, 561
namespaces and, 734
quality of code, 738
size of, 423
Standard C++ versus STL, 561
static, 29
stream, 619–620
using, 735
  windows displaying loaded, 397
libs directory, Boost, 741
-license switch, 754
lifecycle, 287
line numbers, 38
--linewidth switch, 759
-lnk switch, 755
linker, 46, 307, 784, 785, 791
Linux
  Boost library for, 740
  installing Code::Blocks
    graphical installation, 14–16
    standard method, 14
  root directory access and, 649
  support for, 2
list type
  deque type versus, 589
  insert( ) function, 585
overview, 584
  push_back( ) function, 584
  push_front( ) function, 584
  queue type and, 590
  stack type and, 590
Standard Library containers, 685
  using, 353–354, 585–587
  using iterator with, 585
  vector type versus, 588
literals, raw versus cooked, 707–708
LL suffix, 704
LLU suffix, 705
Loaded Libraries window, Code::Blocks IDE, 397
loader, 785
local variables
  in functions, 117–118
  global variables versus, 143
  local_time( ) function, 764
Locale library, 743
locking files, 618
logical operators, 81
logical_and type, 688
logical_not type, 688
logical_or type, 688
Logs & others window, Code::Blocks IDE, 22–23
long double type, 456, 712, 714
long int type, 456, 699, 704–705
loops
arrays and, 241
BOOST_FOREACH macro, 776–778
BOOST_REVERSE_FOREACH macro, 777–778
break statement, 95–96
continue statement, 96–97
defined, 77
do-while loop, 94–95
for loop
  condition for, 87–88
  counting backward, 88–89
  incrementing one step, 89–90
  multiple counter variables, 91–92
  using, 86–87
  using auto keyword, 607
nesting, 97–99
overview, 84–85
in sequence diagrams, 322–323
types of, 85–86
while loop, 92–94
low coupling, 367
lower_bound( ) function, 687
lrand48( ) function, 699–700
LU suffix, 704
lvalue in error messages, 458–459

main( ) function
  command-line arguments, 125–127
  as function, 124
  overview, 36
  return value from, 124–125
make utility
  all option, 793
  clean option, 792
  compiling and linking with, 791
  compiling specific item, 789
defined, 783
different names for, 786
documentation, 793
inference rules and, 787
macros and, 792
Makefiles and, 786
purpose of, 785–786
rules checked before running, 788–789
make_heap( ) function, 687
Makefiles
  all rule in, 793
  backslash in, 793
  clean section, 792
  comments in, 793
  compiling and linking using, 791
  compiling specific item, 789
defined, 129
header files and, 139
implicit rules, 793
inference rules
  multiple dependencies, 789–791
  overview, 787–788
  rules checked before running, 788–789
macros in, 792
modifying for multiple source code files, 135–136
naming, 788
overview, 786
Management window, Code::Blocks IDE, 21–22, 132
manipulators
  creating, 676–680
defined, 669
  formatting output, 637
overview, 673–675

M

-M option, 790
Mac OS X
  Boost library for, 740
  installing Code::Blocks IDE, 13
  root directory access and, 649
  support for, 2
machine code, 784
macros
  BOOST_FOREACH macro, 776–778
  BOOST_REVERSE_FOREACH macro, 777–778
defined, 224
  using in Makefile, 792
mailbox analogy, 250–254
map type
begin( ) function, 574–576
comparing instances, 570–574
end( ) function, 575
instances and, 569
iterating through, 574–577
overview, 512–513
Pair class and, 578
pointers and, 569
set type versus, 581
sort order, 571–572
Standard Library containers, 685
using, 565–566
using with template parameters, 544
math template, 721–722
mathematics
addition, 56–59
division, 62–63
expressions for models, 599
multiplication, 61
operations, 40–43
order of operations, 43
overly large numbers, 43–44
subtraction, 60–61
using parentheses, 44
using variables, 55
math.h library, 104
max( ) function, 687, 689, 700
max_element( ) function, 687
_MAX_PATH constant, 633
mediator class
code example, 361–371
collaborations between objects, 361
overview, 356–360
mem_fun_ref_t type, 688, 690
mem_fun_t type, 688, 690
mem_fun1_ref_t type, 688, 690
mem_fun1_t type, 688, 690
members, class. See also functions
accessing, 188–191
activity diagram and, 287
adjusting access to, 508–510
attributes and, 304
designing base class, 261
designing encapsulation, 256–259
destructors and, 494
drawing static members in UML, 339–340
encapsulation and, 255
explained, 178–180
friend classes and, 520–521
initializing in constructor, 479–484
overloading functions, 199–201
overriding functions, 267–269
overview, 187–188
pointer to member function, 446–449
pointer to static member function, 449
protecting when inheriting class, 262–267
read-only, 255
static
copying instances and, 492
drawing in UML, 339–340
pointer to function, 449
in template, 538–540
templatizing member functions, 559–560
memory
address of function, 673–675
allocating using initializer, 159–160
allocating using new operator, 157–159
defined, 47
dynamic allocation, 157
freeing, 161–163
getting address for variable, 149–150
heap, 146–148
stack, 146–148
using new operator with strings, 160–161
Memory Dump window, Code::Blocks IDE, 396, 436
memory_resource class, 691
merge( ) function, 687
metaclass, 288
metadescription, 288
Metamagical Themas: Questing for the Essence of Mind and Pattern, 288
methodology, and UML, 283, 299
methods. See functions
Metroworks C++, 739
Microsoft Visio, 334
Microsoft Windows
Boost library for, 740
copying files, 664
installing Code::Blocks IDE, 11–13
pathnames, 622
root directory access and, 649
support for, 2
troubleshooting Code::Blocks settings, 18–19
\texttt{min( )} function, 687, 689, 700
\texttt{min} suffix, 713
\texttt{min_element( )} function, 687
\texttt{MinGW}, 389, 397, 561, 790, 792
-\texttt{minmax} switch, 755
\texttt{minus sign (-)}, 40, 261
\texttt{minus} type, 688
\texttt{mismatch( )} function, 687
\texttt{mission-critical applications}, 125
\texttt{mkdir( )} function, 660
-\texttt{MM} option, 791
m..n symbol, 304
modeling classes. \textit{See also} UML
collection of classes as model, 284
design patterns
  Façade pattern, 361
  Observer pattern, 348–356
  overview, 343–344
  Singleton pattern, 344–348
  using mediator class, 356–371
design workflow and, 277
finding missing, 277
overview, 275–276
searching for nouns and verbs, 276
using CRC cards, 276–277
modulus operator, 55, 62
modulus type, 688
\texttt{MPI library}, 743
\texttt{mrand48( )} function, 699–700
\texttt{ms} suffix, 713
\texttt{MS-DOS}, 622
--\texttt{ms-errors} switch, 759
multidimensional arrays
  initializing, 430
  overview, 429–430
  passing, 430–432
\texttt{multimap} type, 685
multiple inheritance, 514–516, 519
multiplication, 40, 61
multiplicities, 304
\texttt{multiplies} type, 688
\texttt{multiset} type, 685\n
\textbf{N}

\texttt{n symbol}, 304
\texttt{name( )} function, 706, 710
\texttt{namespace} keyword, 332
namespaces
  creating from multiple source code files, 473
defined, 471
identifiers and, 331
libraries and, 734
overview, 137
\texttt{std namespace}, 333, 475
in UML diagrams, 332–334
using, 471–473
using from Boost library, 764
variables in, 474–476
naming
  classes, 178, 250
  \texttt{enum} type, 180
  \texttt{Makefiles}, 788
  pointers, 157
  variables, 49, 53–55, 118
\texttt{negate} type, 688
nesting
  classes, 521–526
  loops, 97–99
\texttt{Netbeans}, 25
new operator
  allocating memory using, 157–159, 690
  creating pointer to object, 191
  using \texttt{delete} operator for each, 386
  using with strings, 160–161
newline character, 64–65
\texttt{next_permutation( )} function, 687
\texttt{nmake utility}, 786
\texttt{noboolalpha} flag, 637
\texttt{nocreate} flag, 627–628
\texttt{node}, 310
nonprintable characters, 64–66
non-type parameters, \texttt{template}, 543
--\texttt{no-pretty-print} switch, 759
\texttt{noreplace} flag, 628–629
\texttt{noshowbase} flag, 637
\texttt{noshowpoint} flag, 637
noshowpos flag, 637
noskipws flag, 637
not declared error, 38
not equal (!=) operator, 79
not_equal_to type, 688
notating diagrams, 334–335
nothrow keyword, 596
nouppercase flag, 637
nrand48( ) function, 699
ns suffix, 713
nth_element( ) function, 687
null character (\0), 64
null reference, 465
null value, 163
nullptr, 596
numbers
converting, 218
converting accurately, 773–776
evaluating conditions with, 79–80
floating-point
casting, 466
converting to integers, 216
creating complex numbers, 712
defined, 48
double type, 456
float type, 456
format flags, 634–637
long double type, 456
format flags, 634–637
integers
adding together, 56–59
casting, 466
converting double type to,
773–775
converting to floating-point
type, 216
numbers using Boost library, 741
dividing, 62–63
format flags, 634–637
long int type, 456
mathematics using, 55
max( ) function, 700
min( ) function, 700
multiplying, 61
short int type, 455
subtracting, 60–61
UDL suffixes, 704–705
variables for, 48–50
largest possible, 43–44
precision for output, 637–640
random, 169, 238–240, 698–700
rounding, 775
truncating, 216

.o file extension, 135, 784
.obj file extension, 135, 784
object diagrams, 286, 313–315
object files, 784
Object Management Group (OMG), 282
object-oriented programming. See OOP
objects
aliases, 209–210
attributes and, 249
capabilities explained, 252–253
comparing, 570–574
constructors, 201–202
converting types, 213–218
creating from class, 180–184
defined, 103, 176, 187
designing encapsulation for, 256–259
destructors, 202
capsulation and, 254–256
instance and, 250, 253
mailbox analogy, 250–254
overview, 249–250
passing to function, 194–195
pointers for, 191–194
polymorphism and, 505–508
this pointer, 196–199
watching using breakpoints, 407–408
Observer pattern
automatically adding observer, 354–356
overview, 348–353
Standard Library and, 353–354
Standard Library and, 353–354
oct flag, 636
ofstream class, 620, 623, 624, 632
omanip type, 674–675
OMG (Object Management Group), 282
OOP (object-oriented programming)
C++ history, 59
classes and, 175
encapsulation and, 254–255
mediator class and, 361
operators
   -- operator, 61
   != operator, 79
   && operator, 81
   *= operator, 61
   | | operator, 80
   ++ operator, 58–59, 92
   += operator, 58–59, 69–70
   <= operator, 79
   -= operator, 60
   >= operator, 79
address-of, 445
bang (!), 66, 626
conditional
   Boolean variables and, 73
   numerical conditions, 79–80
   overview, 71–73
delete
   arrays on heap and, 434–435
   for each new operator, 386
   freeing memory, 161–162
delete[], 434, 596
extraction (>>)
   defined, 75
   overloading in class, 670–672
   reading from console, 219
   using with stream, 645–648
insertion (<<)
   defined, 75
   manipulators and, 673–675
   overloading in class, 670–672
   using with stream, 631–632
logical, 81
mathematical, 40
modulus, 55, 62
new
   allocating memory using, 157–159, 690
   creating pointer to object, 191
   using delete operator for each, 386
   using with strings, 160–161
numerical comparison, 79–80
overloading, 238
   s, 710
sizeof, 425
or operator (| |), 80
order of mathematics operations, 43
ostream class, 632
ostream_iterator type, 686
ostringstream type, 169, 214, 728
out flag, 628–629
outer array, 436
outer class, 522–524
output
   to console, 37–40
   cout object, 39
   current working directory, 633
   fields in, 642–643
   formatting, 633–637
   setting width, 640–643
   specifying number precision, 637–640
output_iterator type, 686
output_iterator_tag type, 686
--output-file switch, 759
overflow, 243
overload versus override, 510–511
overloading
   constructors, 478–479
   functions, 120–121, 199–201, 556–559
   operators, 238
   overriding versus, 510–511
   overriding functions, 267–269, 511–514
packages in UML diagrams, 334
Pair class, 578
pair type, 689, 701
Paradigm Plus, 284
parameterized classes, 340–341
parameters
   arguments versus, 112
   command-line, 125–127
   command-line, and arrays, 432–433
   for constructors, 205–206
default, in constructor, 483–484
defined, 109
destructors and, 493
overloading functions, 121
overview, 111, 113
stack and, 412
parameters (continued)

special template
container as, 544
multiple, 544–547
overview, 536, 540
types as, 541–543
values as, 541–543
using, 113–116

parent class, 208
parentheses ( ),
casting types, 218
comparison expressions, 72
if statements, 82
in Makefiles, 792
order of mathematics operations, 44
partial_sort( ) function, 687
partial_sort_copy( ) function, 687
partial_sum( ) function, 687
partition( ) function, 687
Pascal, 324, 414
passing by reference, 166, 171–173, 195
passing by value, 166, 194
passwords, sending as hash, 692
path( ) function, 781
-path_name switch, 755
pathnames, 622, 623
paths, relative versus absolute, 762
pattern matching, 766
peek operation, 590
percent sign (%), 62
persistent classes, 669
phases, RUP
construction phase, 295–296
elaboration phase, 294–295
inception phase, 293–294
overview, 292–293
transition phase, 296–297
Playpux, 14
plus sign (+), 40, 261
plus type, 688
Point structure, 468
pointer_to_binary_function type, 688, 690
pointer_to_unary_function type, 688, 690

pointers
adding and subtracting using array, 244–246
arrays and, 425–428
arrays of, 241–243, 435–437
best practices, 156–157, 162, 173
changing variable using, 151–152
changing what is pointed to, 155–156
declaring multiple, 156
defined, 150
deleting, 192
derreferencing, 151
destructors and, 494
freeing memory, 161–163
to functions, 445–446
to member functions, 446–449
to objects, 191–194
overview, 439–445
passing objects to functions, 195
passing to function, 163–166
references and, 452
returning from function as nonpointer, 170–171
returning from functions, 168–170
to static member functions, 449
steps to using, 193–194
to strings, 153–155
as template parameter, 543
this pointer, 196–199, 446–447
using in Standard Library container
classes, 568–569
vector type and, 577
polling, 349
polyalloc namespace, 691
polymorphic_allocator header, 691
polymorphism
deriving classes, 269–270
destructors and, 495–496
multiple inheritance and, 519
overview, 505–508
pop( ) function, 591
pop operation, 590
pop_front( ) function, 589
pop_heap( ) function, 687
POSIX syntax, 766
posix_time namespace, 764
post-increment, 92
pound sign (#), 66, 69, 261, 793
pow( ) function, 108
power( ) function, 687
precision
  range of numbers versus, 457
  specifying for number output, 637–640
prefixes, UDL, 704–707
pre-increment, 92
preprocessor, Wave, 761
preprocessor directives
  conditional compilation, 225–226
  #define directive, 224–226
  defined, 144
  #else directive, 227
  #endif directive, 226–227
  #ifdef directive, 226–227
  #include directive, 223–224
overview, 223
using, 227–230
prev_permutation( ) function, 687
priority_queue type, 685, 690
private members
  constructor, 346
  derived classes and, 262, 267
  encapsulation and, 255–256
  friend classes and, 520–521
overview, 189
preventing creation of class instances, 525
Singleton pattern, 345
types used within class, 528
ProgramOptions library, 743
project1st type, 688
project2nd type, 688
projects, Code::Blocks IDE
  adding library to, 767–770
  creating, 21, 28–33
  creating from multiple source code files, 133–135
for library
  adding template, 734–735
  creating, 730–732
  setting up, 732–734
  using, 735
opening, 19–20
overview, 28
removing files from, 132
samples, 19–20
splitting into multiple source code files, 130–132
tab for, 22
types of, 29–30
prolog, 415
properties, class
  access methods versus, 259–260
  best practices, 257
protected members
  adjusting access to members, 508–510
  derived classes and, 262
  encapsulation and, 255–256
  friend classes and, 520–521
  inheritance and, 208
  preventing creation of class instances, 525
Singleton pattern, 345
types used within class, 528
protocol
  defined, 646
  reading files, 654
prototypes, function
  in separate source code file, 136–137
  separating class member functions, 185–187
  using, 118–120
  using header files, 138
pseudocode, 325
pseudorandom numbers, 238, 699
ptr prefix, 157
ptr_fun( ) function, 688, 689, 690
public class, 180
public members
  adjusting access to members, 508–510
  best practices, 257
  derived classes and, 262
  encapsulation and, 255
  overview, 189
pure abstract functions, 350
pure virtual functions, 271
push( ) function, 591
push operation, 590
push_back( ) function, 584
push_front() function, 584
push_heap() function, 687
Python, 754, 757
Python library, 743

q suffix, 707
Qt Creator, 25
question mark (?), 72
queue
custom implementation using template, 725–728
defined, 148
queue type, 589–592, 685, 690, 727
QuickBook tool, 750, 758–759
quotient, 62

\r character, 221
rand() function, 169, 239, 699–700
random access, 588, 589, 619
random numbers, 169, 238–240, 698–700
random_access_iterator type, 686
random_access_iterator_tag type, 686
random_sample() function, 687
random_sample_n() function, 687
random_shuffle() function, 687
random-access iterator, 694–696
range of numbers versus precision, 457
ratio type, 712
Rational Rose, 284
Rational Unified Process. See RUP
raw literals, 707–708
raw_storage_iterator type, 686, 690, 691
reading files
extraction operator (>>), 645–648
finding EOF, 648–653
formatted input, 657
getting directory contents, 661–664
reading lines, 653–656
read-only variables, 255
Red Hat Linux, 14
Red Hat Package Manager (RPM), 14
references
catching, 501–502
changing array pointed to, 443
in compiled code, 785
const, 172–173
defined, 171
modifying, 452
passing by, 166, 171–172, 195
passing by const, 172–173
returning from function, 451–453
Standard Library container classes and, 567–568
variables, 450–451
RegEx library
adding library to project, 767–770
defined, 743
documentation for, 766
overview, 766–767
using, 770–772
RegEx_match() function, 766, 771–772
RegEx_search() function, 766, 771–772
registered observers, 352
registers, 412, 415
regression testing, 754
Regression tool, 750, 754
regular expressions. See RegEx library
relationships and UML, 300
relative path, 762
Release tool, 750
release version
defined, 131
location for, 32
reliability of templates, 720
remainder, 55, 62
remove() function, 687, 781
remove_copy() function, 687
remove_copy_if() function, 687
remove_if() function, 687
rename() function, 667
renaming files/directories, 659, 666–667
replace() function, 123, 687
replace_copy() function, 687
replace_copy_if() function, 687
replace_if() function, 687
requirement collection, 289
resources, application, 785
Resources tab, Code::Blocks IDE, 22
return type
from function
calculations in statement, 115
defined, 104
main() function, 124–125
overloading functions, 121
overview, 112, 113
reference, 451–453
struct type, 470–471
for template, 722
void return, 116–117
from lambda expression
specifying, 604–605
using compiler detection, 602–604
return_temporary_buffer() function, 690
reusable code, 344
reverse() function, 687
reverse_bidirectional_iterator type, 686, 690
reverse_copy() function, 687
reverse_iterator type, 686, 690
reverse-assemble, 706
right flag, 636
rm command, 792
e command, 792
rmdir() function, 661
root directory, 649, 762
rope type, 685
rotate() function, 687
rotate_copy() function, 687
RoundEven rounding, 775, 776
rounding numbers, 775
RPM (Red Hat Package Manager), 14
RTTI (Run-time Type Information), 464
Rumbaugh, James, 282
running applications
in Code::Blocks IDE, 33–34, 39
keyboard shortcuts, 46
process overview, 46
Running Threads window, Code::Blocks IDE, 396, 416
runtime
dynamic cast keyword and, 464
library, 160
templates and, 533
types and, 530–531
Run-time Type Information (RTTI), 464
RUP (Rational Unified Process)
construction phase, 295–296
elaboration phase, 294–295
inception phase, 293–294
iterations, 290–292
overview, 288–290
phases overview, 292–293
transition phase, 296–297
rvalue in error messages, 458–459

S

_s operator, 710
s suffix, 713
safe casts, 461
sample projects, 19–20
satisfied condition, 79
SayHello project, 36–40
scalar, 437
--scan switch, 760
scientific flag, 636, 637
scientific notation, 635–636
scope, 339
SD (Secure Digital), 617
search() function, 687
search_n() function, 687
searching in set type, 580–581
second variable, 578
second Clock class, 764
Secure Digital (SD), 617
security of templates, 720
seed48() function, 700
seeding random-number generators, 238
select1st type, 688
select2nd type, 688
selecting text in code editor, 35
semicolon (;)
class definitions, 181
function prototypes, 119
functions and, 113
loop statements, 86
statement syntax, 37, 40
sequence diagrams
comparisons in, 323–324
defined, 286–287
loops in, 322–323
notating, 321
overview, 318–321
sequence_buffer type, 686, 690
sequential access, 619
Serialization library, 743
set type
find() function, 580–581
finding intersection, 583–584
map type versus, 581
overview, 578
Standard Library containers, 685
unionizing, 581–584
using, 579–580
set_difference() function, 687
set_intersection() function, 687
set_symmetric_difference() function, 687
set_union() function, 687
setf() function, 635, 641
setprecision() function, 640, 729
setw flag, 643
SGI (Silicon Graphics, Inc.), 683, 684, 685
Shift key, 35
SHL (shift-left), 419
short int type, 455
shortcut notation (->), 167, 183, 242, 535
showbase flag, 636, 637
showpoint flag, 636, 637
showpos flag, 636, 637
side effects, custom UDL for, 716
Signal Handling window, Code::Blocks IDE, 397
Signals library, 743
signed type, 456–457
significant digits, 638
Silicon Graphics, Inc. (SGI), 683, 684, 685
Simple Mail Transfer Protocol (SMTP), 348
single quote (‘), 64, 66
singleton class, 177, 345
Singleton pattern, 344–348
sizeof operator, 425
skipws flag, 637
slist type, 685
SMTP (Simple Mail Transfer Protocol), 348
sort() function, 611, 687, 702
sort order
map type, 571–572
set type, 580
sort_heap() function, 687
sorting using lambda expression, 610–612
source code files
associating with Code::Blocks IDE, 17
class member definitions, 188
creating project from multiple, 133–135
functions in separate, 136–137
global variables, 142–143
header files
adding only once, 140
brackets versus quotes, 140–141
class definitions, 188
global variables, 142–143
header wrappers, 143–144
header-only libraries in Boost, 742
overview, 138–140
using from Boost library, 764
indentation in, 41
modifying Makefile for, 135–136
namespaces in multiple, 473
order of statements, 51
splitting Code::Blocks project into multiple, 130–132
using multiple, 129–130
SP (stack pointer), 412
spaces, stripping, 382–384
splicing list, 585, 588
square brackets ([ ])
accessing characters in strings, 67
declaring arrays, 424
in lambda expressions, 602
symbol names, 66
using with map type, 513
Index

srand( ) function, 239, 700
srand48( ) function, 700
sstream library, 619
stable_partition( ) function, 687
stable_sort( ) function, 687
stack
constructors and destructors with, 204
debugging using, 412–414
heap versus, 146–148
local variable storage explained, 414–416
overview, 411–412
window in Code::Blocks, 396
stack frame, 396, 415
stack pointer (SP), 412
stack type, 589–592, 685, 690
stakeholders, 294, 323
Standard Library. See also Boost
adaptors, 690
algorithms, 686–687
allocators, 690–691
architecture of, 562
Boost Filesystem library and, 779
categories in, 684–685
comparisons using min( ) and max( ), 700
container classes
comparing instances, 570–574
copy constructor for, 592–594
deque type, 589
equal operator for, 592
instances in, 566–569
iterating through, 574–577
list type, 584–588
listing of, 685
overview, 562
pointers in, 568–569
queue type, 589–592
references and, 567–568
set type, 578–584
stack type, 589–592
unordered_set type, 596–598
defined, 561
documentation, 684
find algorithms, 697–698
function, 687–689
hashes, 691–694
history of, 683
iterators, 685–686
map type, 565–566
no create flag and, 628
Observer pattern and, 353–354
polymorphic allocators, 691
random number generator, 698–700
STL headers versus, 694
STL versus, 561
stream libraries, 620
temporary buffers, 701–702
third-party libraries and, 737
UDLs
basic_string class, 708–710
chrono::duration class, 712–713
complex class, 711–712
using random access iterator, 694–696
utilities, 689
vector type, 562–564
Standard Template Library. See STL
Start Here window, Code::Blocks IDE, 20–21
state, defined, 287–288
state diagrams, 328–329
statechart diagrams, 286–287
statements
assignment, 52
defined, 40
order of, 51
static diagrams, 284–285
static library
adding template, 734–735
creating project, 730–732
defined, 29, 307
setting up project, 732–734
using, 735
static members
copying instances and, 492
drawing in UML, 339–340
pointer to function, 449
in template, 538–540
static_cast keyword, 465–466, 519
status directory, Boost, 742
std namespace, 333, 475, 620
stdio.h header, 659
stepping into/over functions, 392
stereotyping, 308, 315, 336–338
stereotyping symbol (<< >>), 308
STL (Standard Template Library). See also
Boost library; Standard Library
defined, 561
documentation, 684
hash_map type, 692
history of, 683
limitations of, 683
Standard Library headers versus, 694
Standard Library versus, 561
stream libraries, 620
supporting features using GCC
compiler, 689
third-party libraries and, 737
streams
accessing files and, 618–619
to and from class, 670–672
classes and, 669
commonly-used classes, 620–621
defined, 75, 214, 617
file
current working directory, 633
extraction operator (>>), 645–648
fields, 642–643
finding EOF, 648–653
format flags, 634–637
formatting output, 633–634
handling errors when opening files, 624–626
insertion operator (<<), 631–632
ios flags, 627–629
libraries for, 619–620
manipulators, 673–675
opening files, 621–624
precision for numbers, 637–640
reading formatted input, 657
reading lines, 653–656
setting width, 640–643
while loop and, 93
string type, 67
stringify( ) function, 728
strings
accessing characters in, 67
adding onto, 69–70
basic_string class, 708–710
changing part of, 69
class for, 183
combining, 70–71
constant, 70
converting, 214
converting using Boost library, 741
creating tokens from, 772–773
defined, 39
delimiters, 68
functions for, 122–124
getting part of, 68–69
index number for, 385–386
modifying using pointer, 167
numbering positions in, 68
overview, 67
pointing to, 153–155
processing to anticipate bugs, 382–384
random, 169
raw and cooked processing, 707–708
shortcut notation, 167
using BOOST_FOREACH macro with, 778
using new operator with, 160–161
variables for, 48
Stroustrup, Bjarne, 59, 467, 684
struct type
creating custom templates, 722–725
equality between, 469–470
overview, 466–468
packaging other data types as, 468–469
returning from function, 470–471
templates for, 719
subactivities, 327
subscripting, 240
subtraction, 40, 60–61
subtractive_rng type, 688
sudo command, 14
suffixes, UDL, 704–707
SunPro, 739
SUSE Linux, 14
swap( ) function, 687
swap_ranges( ) function, 687
switch statements
break statement, 234
case keyword, 234
default keyword, 234
using, 233–236
wrapping enum type with class, 237
swtiches
  -apple_macro, 755
  -ascii, 755
  -assert_macro, 755
  --boost, 760
  -brief, 756
  -copyright, 755
  -crlf, 755
  -cvs, 756
  --cvs, 760
  --debug, 759
  -deprecated_macro, 755
  -end, 755
  --help, 759
  --indent, 759
  --input-file, 759
  -license, 754
  --linewhit, 759
  -link, 755
  -minmax, 755
  --ms-errors, 759
  --no-pretty-print, 759
  --output-file, 759
  -path_name, 755
  --scan, 760
  -tab, 755
  -text, 756
  --unix-lines, 760
  -unnamed, 755
  --version, 759
symbols, defined, 144
Symbols tab, Code::Blocks IDE, 22
syntax, C++
  anticipating bugs, 385–386
  case sensitivity, 54
  comments, 211–213
  conditional operators, 71–73
  constants, 230–233
  cout object, 39
  endl function, 39
  extraction operator (>>)
    defined, 75
    overloading in class, 670–672
  reading from console, 219
  using with stream, 645–648
indentation, 41
insertion operator (<<)
  defined, 75
  manipulators and, 673–675
  overloading in class, 670–672
  using with stream, 631–632
lambda expressions, 601–605, 609–610
naming variables, 53–55
numbering positions in string, 68
shortcut notation (->), 167, 183, 242, 535
statements
  assignment, 52
  defined, 40
  order of, 51
  strings, 39
  switch statements, 233–236
system( ) function, 660, 665
System library, 743, 778

T
  T placeholder, 534, 556
  tab character, 44–45, 65
  -tab switch, 755
  Targets and Files window, Code::Blocks
    IDE, 397
template keyword, 555
templates
  availability of, 717
  best practices, 719–721
  compilers and, 529
  creating
    class template, 725–728
    customizing for different data types, 728–729
    math template, 721–722
    structure template, 722–725
debbuging errors, 543–544
default parameters, 575
deriving
  class from template, 548–550
  template from class, 550–552
  template from class template, 552–554
  DLLs and, 730
drawing in UML, 340–341
templates (continued)
function
member functions, 559–560
overloading, 556–559
overview, 554–556
history of, 529
instances of, 548
instantiating, 547
library for
adding template code, 734–735
creating project, 730–732
overview, 730
setting up project, 732–734
using, 735
overview, 531–533
parameters for
container as, 544
multiple, 544–547
overview, 536, 540
types as, 541–543
values as, 541–543
separating function code from, 536–538
static members in, 538–540
T placeholder, 534
types of, 719
using, 533–535
using typedef with, 547–548
when to create, 718–719
temporary buffers, 701–702
temporary files, 659
temporary instances, 486–487, 489
temporary_buffer type, 691
testing
Boost library installation, 744–749
Boost test suite, 742
Q/A engineers as testers, 296
regression, 754
workflow for, 275, 296
-text switch, 756
third-party libraries, 738
this pointer, 196–199, 446–447
Thread library, 743
threads, debugging, 416
throw( ) clause, 612
throw statement, 499, 503–504
throwing exceptions
from lambda expression, 612–613
overview, 498–499
rethrowing same exception, 503–504
throwing direct instances, 500–501
tick, 712
tilde (~), 66
time( ) function, 239
time_duration classes, 764
Timer library, 743
tokenizer class, 772–773
tokens, 68, 772
tools directory, Boost, 742, 749
toolset property, 753
top( ) function, 591
top-down design, 102
tracing
assembly code, 417–420
overview, 387–388
transform( ) function, 687, 689
transition phase, 296–297
troubleshooting, Code::Blocks IDE on Windows, 18–19
trunc flag, 628–629
Trunc rounding, 775, 776
truncating numbers, 216
try block
catch block and, 465
in constructor, 493
using, 499–500
type casting
defined, 217
dynamic cast keyword, 461–465
multiple inheritance and, 519
numeric conversions, 218
overview, 460–461
static_cast keyword, 465–466
typedef keyword
advantage of using with pointers, 440–445
creating manipulators, 677
creating map type, 513
custom templates and, 724
declaring inside class, 528
pointer to function, 445–446
using with arrays, 432
using with template, 547–548
typeid( ) function, 706, 729
typedef structure, 706
typename keyword, 540, 543, 544
typeof( ) function, 729
types
auto keyword and, 605
back_insert_iterator type, 686, 690
bidirectional_iterator type, 686
bidirectional_iterator_tag type, 686
binary_compose type, 688, 690
binary_function type, 688
binary_negate type, 688, 690
binder1st type, 688, 690
binder2nd type, 688, 690
bit_vector type, 685
bitset type, 685
bool type, 73–74, 456
casting
defined, 217
dynamic cast keyword, 461–465
multiple inheritance and, 519
numeric conversions, 218
overview, 460–461
static_cast keyword, 465–466
char type, 64, 455, 708, 778
char_producer type, 685
char16_t type, 705, 708
char32_t type, 705, 708
chart_traits type, 689
color-coding on SGI website, 685
const_iterator type, 778
converting, 213–218, 460–461
__Decimal32 type, 707
__Decimal64 type, 707
__Decimal128 type, 707
declaring within classes, 526–528
deque type
overview, 589
queue type and, 590
stack type and, 590
Standard Library containers, 685
vector type versus, 590
divides type, 688
double type
converting to int, 773–775
creating complex numbers, 712
creating conversion UDL, 714
customizing templates for data types, 729
declared, 456
non-standard suffix extensions, 707
precision for, 637–640
pseudorandom numbers, 699
UDL suffixes, 705
dynamic_cast keyword, 461–465
enum type
declaring inside class, 528
drawing in UML, 338–339
naming, 180
overview, 177
wrapping with class, 236–237
equal_to type, 688
float type, 456, 705, 707, 712
__float80 type, 707
__float128 type, 707
forward_iterator type, 686
forward_iterator_tag type, 686
front_insert_iterator type, 686, 690
function return value, 112
greater type, 688
greater_equal type, 688
hash type, 685, 688, 691–694
hash_map type, 692
identity type, 688
input_iterator type, 686
input_iterator_tag type, 686
insert_iterator type, 686, 690
int type, 49, 455, 704–705, 773–775
integers
adding together, 56–59
casting, 466
converting double type to, 773–775
converting to floating-point numbers, 216
converting using Boost library, 741
dividing, 62–63
format flags, 634–637
types (continued)

mathematics using, 55
max( ) function, 700
min( ) function, 700
multiplying, 61
random number as, 699
subtracting, 60–61
UDL suffi
xes, 704–705
variables for, 48–50

istream_iterator type, 686
istringstream type, 214
iterator type, 574, 581, 773, 778
iterator_traits type, 686
less type, 688
less_equal type, 688
list type
ddeque type versus, 589
insert( ) function, 585
overview, 584
push_back( ) function, 584
push_front( ) function, 584
queue type and, 590
stack type and, 590
Standard Library containers, 685
using, 353–354, 585–587
using iterator with, 585

vector type versus, 588
logical_and type, 688
logical_not type, 688
logical_or type, 688
long double type, 456, 712, 714
long int type, 456, 699, 704–705
map type
begin( ) function, 574–576
comparing instances, 570–574
end( ) function, 575
instances and, 569
iterating through, 574–577
overview, 512–513
Pair class and, 578
pointers and, 569
set type versus, 581
sort order, 571–572
Standard Library containers, 685
using, 565–566
using with template parameters, 544

mem_fun_ref_t type, 688, 690
mem_fun_t type, 688, 690
mem_fun1_ref_t type, 688, 690
mem_fun1_t type, 688, 690
minus type, 688
modulus type, 688
multimap type, 685
multiplies type, 688
multiset type, 685
negate type, 688
not_equal_to type, 688
omanip type, 674–675
ostream_iterator type, 686
ostringstream type, 169, 214, 728
output_iterator type, 686
output_iterator_tag type, 686
pair type, 689, 701
plus type, 688
pointer_to_binary_function type,
688, 690
pointer_to_unary_function type,
688, 690
priority_queue type, 685, 690
project1st type, 688
project2nd type, 688
queue type, 589–592, 685, 690, 727
random_access_iterator type, 686
random_access_iterator_tag type, 686
ratio type, 712
raw_storage_iterator type,
686, 690, 691
reverse_bidirectional_iterator
type, 686, 690
reverse_iterator type, 686, 690
rope type, 685
runtime and, 530–531
select1st type, 688
select2nd type, 688
sequence_buffer type, 686, 690
set type
find( ) function, 580–581
finding intersection, 583–584
map type versus, 581
overview, 578
Standard Library containers, 685
unionizing, 581–584
using, 579–580
short int type, 455
signed type, 456–457
slist type, 685
stack type, 589–592, 685, 690
static_cast keyword, 465–466
string type, 67
strings
accessing characters in, 67
adding onto, 69–70
basic_string class, 708–710
changing part of, 69
class for, 183
combining, 70–71
constant, 70
converting, 214
converting using Boost library, 741
creating tokens from, 772–773
defined, 39
delimiters, 68
functions for, 122–124
getting part of, 68–69
index number for, 385–386
modifying using pointer, 167
numbering positions in, 68
overview, 67
pointing to, 153–155
processing to anticipate bugs, 382–384
random, 169
raw and cooked processing, 707–708
shortcut notation, 167
using BOOST_FOREACH macro with, 778
using new operator with, 160–161
variables for, 48
struct type
creating custom templates, 722–725
equality between, 469–470
overview, 466–468
packaging other data types as, 468–469
returning from function, 470–471
templates for, 719
subtractive_rng type, 688
as template parameter, 543
temporary_buffer type, 691
typeinfo structure, 706

unary_compose type, 688, 690
unary_function type, 688
unary_negate type, 688
unordered_map type, 692
unordered_set type
C++ 11 extensions, 597
overview, 596–597
using, 597–598
unsigned type, 154, 456–457, 704–705
variable, 49
vector type
array versus, 563
begin( ) function, 574–576
defined, 536
deque type versus, 589, 590
defend( ) function, 575
iterating through, 574–577
list type versus, 588
size of, 577
stack type and, 590
Standard Library containers, 685
using, 562–564
using random access iterator, 694–696
void type, 456, 458
wchar_t type, 456, 705, 708, 778

U
prefix, 705
U suffix, 704
Ubuntu, 14
UDLs (User-Defined Literals)
basic_string class, 708–710
C++ 11 extensions, 703
C++ 14 support, 709
chrono::duration class, 712–713
complex class, 711–712
custom
conversion, 714–715
for side effects, 716
type, 715
overview, 703
prefixes, 704–707
raw versus cooked literals, 707–708
suffixes, 704–707
uses for, 703–704
UL suffix, 704
ULL suffix, 705
UML (Unified Modeling Language)
  activity diagrams, 326–328
  capabilities of, 283
  CASE tools and, 284
  class diagrams
    aggregation, 305–307
    composition, 303, 305–307
    displaying inheritance, 302, 304–305
    enumerations, 338–339
    overview, 300–301
    static members, 339–340
    templates, 340–341
  collaboration diagrams, 325–326
  colons in, 324
  component diagrams, 307–310
  defined, 279
  deployment diagrams, 310–311
  design workflow, 274–275
  diagrams available for, 284
  dynamic diagrams, 284, 286–287
  extending, 336–338
  flowcharts and, 280–281
  history of, 279
  importance of, 297
  iterations, 290–292
  lifecycle, 287
  metadescription, 288
  methodology versus, 283, 299
  multiplicities, 304
  namespaces in diagram, 331–334
  notating diagrams, 334–335
  object diagrams, 313–315
  packages in, 334
  relationships and, 300
  RUP using, 288–290
  sequence diagrams
    comparisons in, 323–324
    loops in, 322–323
    notating, 321
    overview, 318–321
  shorthand notation, 261–262
  specification, 331
  state and, 287–288
  state diagrams, 328–329
  static diagrams, 284–285
  stereotyping, 308
  tags in, 335
  use case diagrams
    creating, 315–316
    defining on paper, 317–318
    event flow example, 317
    matching with requirements, 318
    workflow using, 288–290
  unary_compose type, 688, 690
  unary_function type, 688
  unary_negate type, 688, 690
  underlined text in diagram, 313
  underscore (_), 55
Unified Modeling Language. See UML
uninitialized_copy_n( ) function, 687
unionizing set types, 581–584
unique( ) function, 687
unique_copy( ) function, 687
unbuf flag, 636
uninitialized_copy( ) function, 687, 690
uninitialized_copy_n( ) function, 690
uninitialized_fill( ) function, 687, 690
uninitialized_fill_n( ) function, 687, 690
Universal Serial Bus (USB), 617
--unix-lines switch, 760
-unnamed switch, 755
unordered_map type, 692
unordered_set type
  C++ 11 extensions, 597
  overview, 596–597
  using, 597–598
unseat reference, 452
unsetf( ) function, 634
unsigned type, 154, 456–457, 704–705
upper_bound( ) function, 687
uppercase flag, 636, 637
us suffix, 713
USB (Universal Serial Bus), 617
use case diagrams
  creating, 315–316
  defining on paper, 317–318
  event flow example, 317
matching with requirements, 318
overview, 286
User-Defined Literals. See UDLs
using statement, 472–474, 508, 620

V
validation, input, 766
value_type( ) function, 686
values
  passing by, 166, 194
  as template parameter, 543
  of variables, 49
variables
  accessing, 50
  adding integers, 56–59
  allocating memory
    using initializer, 159–160
    using new operator, 157–159
aspects of, 48
assignment, 50
auto keyword and, 605
Boolean, 73–74
breakpoints and, 404
changing using pointer, 151–152, 163–166
character, 64–66
constants, 230–233
copying, 52–53
creating, 48–49
declaring multiple, 51
derstructors and, 494
dividing integers, 62–63
eapsulation and, 255
global, 142–143
initializing, 53
integer, 48–50
local, 117–118
lvalue in error messages, 458–459
mathematics using, 55
member, 179, 187
memory address for, 146, 149–150
multiplying integers, 61
namespace and, 474–476
naming, 53–55
overview, 47–48
passing to function, 107–109
pointer, 150
read-only, 255
reference, 450–451
rvalue in error messages, 458–459
saving function result as, 108
setting value equal to another, 52–53
setting values, 51–52
stack, 204
stack and, 412, 414–416
string, 67–71
subtracting integers, 60–61
testing different values for, 408–409
type casting, 460–461
watching using breakpoints, 406–407
VB.NET, 259
VC++.
  See Visual C++
vector type
  array versus, 563
  begin( ) function, 574–576
defined, 536
deque type versus, 589, 590
end( ) function, 575
iterating through, 574–577
list type versus, 588
size of, 577
stack type and, 590
Standard Library containers, 685
using, 562–564
using random access iterator, 694–696
--version switch, 759
virtual functions
  abstract, 271, 305
dynamic cast keyword and, 463
polymorphism, 270
pure, 271
virtual inheritance, 516–520
virtual keyword
  constructors and, 478
deriving class using, 516–517
dstructors and, 495–497
overrideing member functions, 269
Visual C++
  Boost support, 739
debugger, 397
dynamic cast keyword and, 464
Visual C++ (continued)
  header files and, 139
  lvalue and rvalue expressions, 458
  make utility and, 783
  Standard Library and, 561
  std namespace and, 475
  version 14 support, 709
Visual Studio, 740, 759, 761
Vlissides, John, 343
void return value for functions, 116–117
void type, 456, 458

w
w suffi
x, 707
Watches window, Code::Blocks IDE, 396
Wave library, 743, 750, 761
wchar_t type, 456, 705, 708, 778
while loop
  break statement, 95–96
  checking for EOF, 653
  continue statement, 96–97
  nesting, 97–99
  overview, 85–86
  using, 92–94
white space, 647
wide character set, 456
wiki-style documentation, 750, 758
windows.h header, 664
workflow
  building applications, 274–275
  using UML, 288–290
wrapping enum type, 236–237
WriteFile( ) function, 649
writing files
  current working directory, 633
  fields in, 642–643
  format flags, 634–637
  formatting output, 633–634
  insertion operator (<<), 631–632
  setting width, 640–643
  specifying number precision, 637–640
wxWidgets plugin, 22

X
X Windowing system, 30
Xcode, 13
XSL (eXtensible Stylesheet Language), 750, 757

Yellow Dog Linux, 14