

# Index

## SYMBOLS

&= (ampersand, equal) operator, 8  
 & (ampersand) operator, 8  
 \ (backslash) escape character, 4  
 ^= (caret, equal) operator, 9  
 ^ (caret) operator, 9  
 \r (carriage return) escape character, 4  
 {} (curly braces), 153  
 /= (division, equal) operator, 8  
 / (division) operator, 8  
 -- (double minus) decrement operator, 8  
 = (equals) assignment operator, 8, 264  
 ! (exclamation point) operator, 8  
 %= (mod, equal) operator, 8  
 % (mod) operator, 8  
 \*= (multiplication, equal) operator, 8  
 \* (multiplication) operator, 8  
 \n (new line) escape character, 4  
 ( ) parentheses in code, 154  
 | (pipe) operator, 8  
 + (plus) operator, 8  
 += (plus, equal) operator, 8  
 ++ (plus, plus) operator, 8  
 \" (quotation mark) escape character, 4  
 :: (scope resolution) operator, 161, 237–239  
 -= (subtraction, equal) operator, 8  
 - (subtraction) operator, 8  
 \t (tab) escape character, 4  
 | (vertical line) operator, 8

## A

*The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)*, Joel Spolsky, 805

### abstract classes

abstract base classes, 242–243  
 implementations, 218–221  
 interfaces, 218–221

### abstraction

code reuse, 107–108  
 defined, 47–48  
 designing, 76  
 interface versus implementation, 73

### access control

members (classes), 159–160  
 methods, 159–160  
 private, 159–160  
 protected, 159  
 public, 159

### access specifiers, 159–160

### accessing

data members (classes), 161, 195–196  
 out-of-bounds memory, 378

### accessing fields of object elements, 573–574

accumulate() algorithm, 97, 623–624

### ad hoc comments, 144

### adapter design pattern, 768–773

adapters. *See* function object adapters

add() method, 209

adjacent\_difference() algorithm, 97

adjacent\_find() algorithm, 96, 633

*Advanced CORBA Programming with C++*, Michi Henning and Steve Vinoski, 810

### aggregation, 63, 110

Alexandrescu, Andrei, *Modern C++ Design: Generic Programming and Design Patterns Applied*, 811

### algorithms

accumulate(), 97, 623–624  
 adjacent\_difference(), 97  
 adjacent\_find(), 96, 633  
 binary\_search(), 96, 635  
 choosing, 100  
 comparison, 97, 635  
 copy(), 98, 640–641  
 copy\_backward(), 98  
 count(), 97, 635

## algorithms (continued)

- count\_if(), 97, 635
  - defined, 95, 620
  - equal(), 97, 635–636
  - equal\_range(), 96, 635
  - fill(), 98
  - fill\_n(), 98
  - find(), 96, 620–622, 633
  - find\_end(), 96
  - find\_first\_of(), 96, 633
  - find\_if(), 96, 622–623, 633
  - for\_each(), 98, 637–639
  - generate(), 98
  - generate\_n(), 98
  - includes(), 99, 646
  - inner\_product(), 97
  - inplace\_merge(), 99
  - iterator traits, 662
  - iterators, 95, 631–632
  - iter\_swap(), 98
  - lexicographical\_compare(), 97, 635–637
  - lists, 586–587
  - lower\_bound(), 96, 635
  - make\_heap(), 99, 645
  - max(), 96, 632–633
  - max\_element(), 96, 633
  - merge(), 99, 644
  - min(), 96, 632–633
  - min\_element(), 96, 633
  - mismatch(), 97, 635–636
  - modifying, 98, 639
  - next\_permutation(), 98
  - nonmodifying, 96, 633
  - nth\_element(), 99, 645
  - numerical processing, 97, 635
  - operational, 98, 637
  - partial\_sort(), 99, 645
  - partial\_sort\_copy(), 99
  - partial\_sum(), 97
  - partition(), 99, 645
  - pop\_heap(), 99, 645
  - prev\_permutation(), 98
  - push\_heap(), 99, 645
  - random\_shuffle(), 99, 645
  - remove(), 98, 642
  - remove\_copy(), 98, 642
  - remove\_copy\_if(), 98, 642
  - remove\_if(), 98, 642
  - replace(), 98, 641
  - replace\_copy(), 98
  - replace\_copy\_if(), 98
  - replace\_if(), 98, 641
  - reverse(), 98, 643
  - reverse\_copy(), 98, 643
  - rotate(), 98
  - rotate\_copy(), 98
  - search(), 96, 633
  - search\_n(), 96
  - set\_difference(), 99, 646
  - set\_intersection(), 99, 646
  - set\_symmetric\_difference(), 99, 646
  - set\_union(), 99, 646
  - sort(), 99, 643–644
  - sort\_heap(), 99, 645
  - sorting, 99
  - stable\_partition(), 99
  - stable\_sort(), 99, 643
  - swap(), 96, 632
  - swap\_ranges(), 98
  - transform(), 98, 639–640
  - unique(), 98, 643
  - unique\_copy(), 98
  - upper\_bound(), 96, 635
  - utility, 96, 632–633
  - writing, 660–662
- aliasing, 102**
- Allison, Chuck, *Thinking in C++, Volume 2: Practical Programming (Second Edition)*, 804**
- allocators, 656, 682**
- Altova Software xmlspy, 723, 810**
- ambiguity of names and multiple inheritance, 249–252**
- ambiguous base classes, 249–252**
- ampersand, equal (&) operator, 8**
- ampersand (&) operator, 8**
- API (application programming interface), 74–75, 78**
- Applied C++: Practical Techniques for Building Better Software*, Philip Romanik and Amy Muntz, 805**
- arguments**
- operator overloading, 434
  - variable-length argument lists, 345–347
- arithmetic function objects, 624–625**
- arithmetic operator overloading, 212–215, 438–439**
- arrays**
- associative arrays, 93
  - containers, 611–612
  - declaring, 15
  - defined, 15–16, 353
  - deleting, 20, 356–357
  - dynamic allocation, 19–20
  - heap-based arrays, 354
  - multidimensional, 16
  - multidimensional arrays, 357–360
  - non-integral array indices, 447–448
  - objects, 355–356
  - one-dimensional, 16
  - pointers, 362–364
  - realloc() function, 355
  - stack-based arrays, 354
  - variable-sized arrays, 355
- arrow operator, 452**
- The Art and Science of C: A Library Based Introduction to Computer Science*, Eric S. Roberts, 806**
- The Art of Computer Programming Volume 1: Fundamental Algorithms (Third Edition)*, Donald E. Knuth, 807**
- The Art of Computer Programming Volume 2: Seminumerical Algorithms (Third Edition)*, Donald E. Knuth, 807**

**The Art of Computer Programming Volume 3: Sorting and Searching (Third Edition), Donald E. Knuth, 807**

asm keyword, 504

assembly code, 504–505

assert macro, 540–541

assigning object values, 177–178

assignment operator, 8, 264

code example, 191–193

containers, 563

declaring, 178–179

defining, 179–180, 194

disallowing assignment, 194

distinguishing from copy constructors, 180

vectors, 571

associative arrays, 93

associative containers, 595, 683–687

at() method, 568

attributes (XML), 711

auditing voter registrations example program, 648–653

auto\_ptr template, 88–89

**B**

back() method, 568

backslash (\) escape character, 4

bad() method, 383

base classes

abstract, 242–243

ambiguous, 249–252

virtual, 269–270

Beck, Kent

eXtreme Programming eXplained, 128, 808

Refactoring: Improving the Design of Existing Code, 808

begin() method, 565

behaviors, 59–60

BeOS framework, 750

bidirectional iterators, 564

bidirectional streams, 396–397

big-O notation, 82–83

binary logical operator overloading, 441

binary operators, 8

binary\_search() algorithm, 96, 635

binders, 627–628

bitsets, 93–94

bitwise operator overloading, 441

black box testing, 507

Boehm, Barry W., *A Spiral Model of Software Development and Enhancement*, 807

bool variable type, 7

boolalpha manipulator, 383, 389

Boolean expression conversions, 455–457

braces {}, 153

Brant, John, *Refactoring: Improving the Design of Existing Code*, 808

buffer overflow errors, 378

buffers (output streams), 382

**bugs**

buffer overflow errors, 378

catastrophic bugs, 528

error logging, 528–530

Fundamental Law of Debugging, 527

life cycle, 508–509

noncatastrophic bugs, 528

nonreproducible, 543–544

regression testing, 525

reproducible, 541–543

root cause, 528

tips for avoiding bugs, 528

**Bugzilla bug-tracking tool, 509–510****Bulka, Dov, *Efficient C++: Performance Programming Techniques*, 809**

bundling third-party applications, 85

**C**

C functions, 630

*C Pocket Reference*, Peter Prinz, Tony Crawford (Translator), Ulla Kirch-Prinz, 806*The C Programming Language (Second Edition)*, Brian W. Kernighan and Dennis M. Ritchie, 806

C standard library, 88

C++ FAQ LITE, Marshall Cline, 804

C++ FAQs (Second Edition), Marshall Cline, Greg Lomow, and Mike Giru, 804

C++ Gotchas, Stephen C. Dewhurst, 804

C++ *How to Program (Fourth Edition)*, Harvey M. Deitel and Paul J. Deitel, 803C++ *in a Nutshell*, Ray Lischner, 804C++ *Primer Plus*, Stephen Prata, 804C++ *Primer (Third Edition)*, Stanley B. Lippman and Josée Lajoie, 803*The C++ Programming Language (Special Third Edition)*, Bjarne Stroustrup, 1, 804

C++ Resources Network Web site, 805

*The C++ Standard: Incorporating Technical Corrigendum No. 1* (John Wiley & Sons), 804

C++ standard library, 87, 89

*The C++ Standard Library: A Tutorial and Reference*, Nicolai M. Josuttis, 805

C++ strings, 21–22

C++ *Templates: The Complete Guide*, David Vandevoorde and Nicolai M. Josuttis, 806

cache invalidation, 473

caching, 472–473

calling

member functions, 629–630

methods, 161–163

capitalization of code, 150

caret, equal (^=) operator, 9

caret (^) operator, 9

carriage return (\r) escape character, 4

## casting

- downcasting, 239–240
- pointers, 361–362
- slicing, 239
- upcasting, 239
- variables, 7–8

## casts

- `const_cast`, 338–339, 342
- `dynamic_cast`, 267–268, 341–342
- `reinterpret_cast`, 340, 342
- `static_cast`, 339–340, 342

## catastrophic bugs, 528

## catching exceptions, 402, 405–406, 732–733

## centrality of distributed computing, 694

## Cerami, Ethan, *Web Services Essentials*, 810

## chain of responsibility design pattern, 776–778

## changing method characteristics, 253–256

## char variable type, 7

## character sets

- facets, 400
- internationalization, 397–398
- locales, 398–400
- non-Western, 398
- Unicode, 398
- wide characters, 397–398

## Chess program, 51–55

## child classes, 234. *See also* subclasses

## cin input stream, 384–385

## class templates. *See* template classes

## classes

- abstract classes
  - abstract base classes, 242–243
  - implementations, 218–221
  - interfaces, 218–221
- access specifiers, 159–160
- base classes
  - abstract, 242–243
  - ambiguous, 249–252
  - virtual, 269–270
- child classes, 234
- code example, 730–731
- constructors, 27
- data members
  - access control, 159–160
  - accessing, 161, 195–196
  - `const`, 196–198
  - `const` reference, 199
  - defined, 158
  - pointers, 217–218
  - reference, 198–199
  - `static`, 195–196, 333
  - `static const`, 197–198
- declaring, 26–27
- defined, 26, 58
- definition, 158, 730–731
- destructors, 27
- exception classes, 419–421, 732
- extending, 224–225, 731–732
- friend, 208
- `fstream`, 396
- `ifstream`, 392
- implementation, 731
- instances, 58–59
- `iostream`, 391
- methods
  - access control, 160
  - calling, 161–163
  - defined, 26–29, 158
  - defining, 161
  - `static`, 333
  - `this` pointer, 163–164
- mix-in classes
  - defined, 73
  - designing, 747–749
  - implementation, 749
  - uses, 749
  - `XMLSerializable` class, 723–724
- nested, 206–207
- `ofstream`, 392
- order of declarations, 160
- `ostream`, 391
- pair class, 595–596
- parent classes
  - constructors, 234–235
  - destructors, 235–237
  - inheritance, 224–225
- reusable code, 78
- `string`, 88, 367–369
- `stringstream`, 397
- subclasses
  - adding functionality, 231–233
  - copy constructors, 263–264
  - creating, 731–732
  - default arguments, 260–261
  - assignment (=) operator, 264
  - implementation, 732
  - inheritance, 226–227
  - overridden methods, 230
  - replacing functionality, 233
  - template classes, 293–294
- superclasses
  - default arguments, 260–261
  - defined, 65
  - inheritance, 224–225
- template classes
  - method definitions, 277–279, 281–282
  - subclasses, 293–294
  - syntax, 276–277
  - uses, 273
  - writing, 273–275, 734–736
- `wifstream`, 397
- `wofstream`, 397

## cleanup (exceptions), 423

## clear() method, 383

**Cline, Marshall**

C++ FAQ LITE, 804

C++ FAQs (Second Edition), 804

**code bloat, 111, 281****code coverage (unit testing), 511****Code Reading: The Open Source Perspective, Diomidis Spinellis, 808****code reuse**

abstraction, 107–108

advantages, 79

aggregation, 110

capabilities, 81

categories of, 78

disadvantages, 79–80

error checking, 112

frameworks, 78

goals, 106–107

hierarchies, 109–110

inheritance, 230

interfaces, 107–108, 112–118

libraries, 78

licensing, 84

limitations, 81

open-source libraries, 86–87

performance, 81

philosophy, 106

platforms, 83–84

polymorphism, 66

program design, 49–50

safeguards, 112

stand-alone functions or classes, 78

subsystems, 109

support, 84–85

templates, 110–112

tips for writing, 49–50, 108–109

**coding**

const keyword, 151

constants, 151

custom exceptions, 152

decomposition, 145–148

documentation, 136–145

formatting, 152–154

importance of “good” code, 135–136

naming conventions, 148–151

references versus pointers, 151–152

stylistic consistency, 155

**coercing variables, 7****collections, 60****commands. See specific commands by name****comments**

ad hoc, 144

complicated code, 137–138

C-style, 2

defined, 2

fixed-format, 142–143

interfaces, 115–116

line by line, 140–141

metainformation, 139

prefix, 141–142

self-documenting code, 144–145

testing, 526

usage information, 136–137

**Common Object Request Broker Architecture (CORBA).****See CORBA (Common Object Request Broker Architecture)****comparing vectors, 571–572****comparison algorithms, 97, 635****comparison function objects, 625–626****comparison operator overloading, 215–216****compiler support for templates, 272****compiler-generated constructors, 175–176****compile-time debug mode, 530–532****component interface, 75****components, 59****Computer Architecture: A Quantitative Approach (Third Edition), John L. Hennessy and David A. Patterson, 809****Computer Organization & Design: The Hardware/Software Interface (Second Edition), David A. Patterson and John L. Hennessy, 809****conditional operators, 13–14****conditionals**

defined, 12

if/else statements, 12

switch statements, 12–13

**console streams, 380****const data members (classes), 196–198****const keyword**

constants, 25

methods, 200–202, 333

pointers, 330–332, 362

proper use, 151

references, 26, 199, 332

uses, 25

variables, 25, 330

**constants**

const keyword, 25

defining, 25

smart constants, 150

uses, 151

**const\_cast, 338–339, 342****const\_iterator, 574****const\_iterators, 565****constructors**

child classes, 234

compiler-generated, 175–176

default constructor, 168–171

defined, 27, 165

error handling, 427–428

on the heap, 167

initializer lists, 171–172

multiple constructors, 167–168

parent classes, 234–235

on the stack, 166–167

vectors, 569–570

writing, 166

## containers

- accessing fields of object elements, 573–574
- arrays, 611–612
- assignment operator, 563
- associative, 595, 683–687
- bitsets, 93
- copy constructor, 563
- defined, 89–90
- deque, 91, 565, 584
- destructors, 563
- element requirements, 562–563
- error checking, 563
- exceptions, 563
- hash tables
  - accessor operations, 690–691
  - constructors, 687
  - erase operations, 688–689
  - insertion operations, 687–688
  - methods, 672–675
  - typedefs, 671–672
  - writing, 662–670
- iterators, 95, 564–565
- lists, 90–91, 565, 584
- maps, 93, 596–599
- multimaps, 93, 604–605
- multisets, 92–93, 610–611
- performance comparison, 94
- pointers, 563
- priority queues, 91–92, 591–592
- queues, 91, 588–589
- reference semantics, 562
- reversible, 682
- sequential, 691
- sequential containers, 565
- sets, 92–93, 608
- smart pointers, 563
- stacks, 92
- std namespace, 562
- streams, 613–618
- strings, 612–613
- value semantics, 562
- vectors, 90, 565–566
- writing, 662–670

**conversion operators, 453–455**

**conversions (implicit and explicit), 211**

**converting variable types, 7**

`copy()` **algorithm, 98, 640–641**

**copy constructors**

- calling, 173–174
- containers, 563
- defined, 172
- defining, 190–191
- object members, 181
- private, 194
- subclasses, 263–264

`copy_backward()` **algorithm, 98**

`copyFrom()` **method, 193–194**

**copying vectors, 571**

**CORBA (Common Object Request Broker Architecture)**

- client process, 708–709
- defined, 702
- Interface Definition Language (IDL), 702–705
- intermachine communication, 706
- Internet Inter-ORB Protocol (IIOP), 702
- interprocess communication, 706
- location transparency, 702
- nameserver, 706
- Object Request Broker (ORB) framework, 706
- omniORB framework, 702
- open-source implementations, 702
- Portable Object Adapter (POA), 704
- server process, 707–708
- Web site, 810

**Cormen, Thomas H., *Introduction to Algorithms (Second Edition)*, 807**

`count()` **algorithm, 97, 635**

**counters, 149**

`count_if()` **algorithm, 97, 635**

`cout` **output stream, 381**

**covariant returns types, 253**

**cppunit open-source unit testing framework, 516–523**

**creating**

- mix-in classes, 747–749
- objects, 164–166
- subclasses, 731–732

**cross-language applications**

- C and C++, 494–498
- C++ with assembly code, 504–505
- C++ with perl and shell scripts, 501–502
- Java and C++ with JNI, 499–501

**cross-platform applications**

- architecture issues
  - binary compatibility, 490
  - cross-compiling, 490
  - open source distribution, 490
  - word and type sizes, 490–491
  - word order, 491–492
- code reuse, 83–84
- implementation issues
  - compilers, 492–493
  - libraries, 493
- platform-specific features, 493–494

**cruff, 147**

**C-style comments, 2**

**C-style strings, 21, 365–366**

**Cunningham and Cunningham, *The Portland Pattern Repository*, 811**

**curly braces {}, 153**

**custom exceptions, 152**

**customizability of interfaces, 117**

**D****Darwin, Ian F., *Java Cookbook*, 806****data members (classes)**

- access control, 159–160
- access specifiers, 159–160
- accessing, 161, 195–196
- const, 196–198
- const reference, 199
- declaring, 158
- defined, 26
- pointers, 217–218
- reference, 198–199
- static, 195–196, 333
- static const, 197–198

**data members (references), 326****Death March, Edward Yourdon, 132, 808****debug traces**

- debug modes, 530–535
- defined, 530
- ring buffers, 535–540

**debuggers**

- Gnu Debugger (gdb), 556–558, 809
- Rational Purify, 809
- symbolic debugger, 542
- Valgrind, 809

**debugging**

- Article Citations example, 548–559
- assert macro, 540–541
- error logging, 528–530
- Fundamental Law of Debugging, 527
- memory errors, 544–547
- multithreaded programs, 547
- nonreproducible bugs, 543–544
- reproducible bugs, 541–543
- tips for avoiding bugs, 528

dec **manipulator, 383, 389****declarations**

- arrays, 15
- classes, 26–27
- data members (classes), 158
- functions, 16–17
- methods, 158
- order of, 160
- pointers, 19
- variables, 6

**decomposition, 145–148****decorator design pattern, 773–776****decrement operator, 8**

- overloading, 439–441

**default constructor, 168–171****default parameters, 203–204**#define [key] [value] **preprocessor directive, 3****defining**

- constants, 25
- methods, 161

**Deitel, Harvey M. and Paul J., *C++ How to Program (Fourth Edition)*, 803**delete **command, 20****delete expression, 457–458**delete **keyword, 351****deleting arrays, 20, 356–357****deques, 91, 94, 584****dereferencing operator overloading, 449–451****dereferencing pointers, 20–21****Derge, Gillmer J., *STL Tutorial and Reference Guide (Second Edition)*, 805****deserialization**

C++, 697

XML, 717

**design patterns**

- adapter, 768–773
- chain of responsibility, 776–778
- decorator, 773–776
- defined, 51, 102
- factory, 760–766
- iterator pattern, 103
- observer, 778–781
- proxy, 766–768
- singleton, 754–760

**Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, 811****design-level efficiency**

- cache invalidation, 473
- caching, 472–473
- defined, 466
- object pools, 473–478
- thread pools, 479

**destructors**

- containers, 563
- error handling, 428
- freeing memory, 186
- non-virtual destructors, 266–267
- objects, 27, 176–177
- parent classes, 235–237
- vectors, 569–570
- virtual destructors, 236–237

**Dewhurst, Stephen C., *C++ Gotchas*, 804****disabling support for language features, 472****disallowing**

- assignment, 194
- pass-by-value, 194

**distributed computing**

- centrality, 694
- content, 695
- grid computing, 694
- load balancer, 694
- reliability, 694
- rendering farms, 694
- scalability, 693–694
- SETI@home project, 694
- versus networked computing, 695–696

## distributed objects

- CORBA, 702–709
  - deserialization, 697
  - marshalling, 696–700
  - remote procedure call (RPC), 700–701
  - serialization, 696–700
  - unmarshalling, 697
  - XML, 712, 723, 725–726
- division, equal (/=) operator, 8**
- division (/) operator, 8**
- Document Object Model (DOM) parser, 717**
- Document Type Definition (DTD) (XML), 721–722**
- documentation**

- comments, 136–145
- Doxygen tool, 142–143
- interfaces, 115–116
- JavaDoc tool, 142
- performance, 83
- self-documenting code, 145

## DOM (Document Object Model) parser, 717

## double deletion (memory leaks), 377

## double dispatch, 741, 744–747

## double minus (--) decrement operator, 8

## double variable type, 7

## doubly linked lists, 90–91

## do/while loops, 15

## downcasting, 239–240

## Doxygen, Dimitri van Heesch, 808

## Doxygen documentation tool, 142–143

## DTD (Document Type Definition) (XML), 721–722

## dumb pointers, 565

## Dustin, Elfriede, *Effective Software Testing: 50 Specific Ways to Improve Your Testing*, 809

## dynamic memory

- advantages of using, 350
- allocation, 183–190, 351–352
- arrays, 353–360
- deallocation, 351–352
- defined, 18–19
- failure of memory allocation, 353
- free() function, 353
- malloc() function, 352–353
- mental model, 350–351
- orphaned memory, 351–352
- realloc() function, 355

## dynamic\_cast, 267–268, 341–342

## dynamic-length vectors, 568–569

# E

## Eckel, Bruce

- Thinking in C++, Volume 1: Introduction to Standard C++ (Second Edition)*, 803
- Thinking in C++, Volume 2: Practical Programming (Second Edition)*, 804
- Effective C++ (Second Edition): 50 Specific Ways to Improve Your Programs and Designs*, Scott Meyers, 804

## *Effective Software Testing: 50 Specific Ways to Improve Your Testing*, Elfriede Dustin, 809

## *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*, Scott Meyers, 805

## efficiency (of programs)

- defined, 465–466
- design-level, 466, 472–479
- exceptions, 471
- inline functions, 472
- inline methods, 472
- language-level, 466–472
- RTTI, 471–472
- virtual methods, 471

## *Efficient C++: Performance Programming Techniques*, Dov Bulka and David Mayhew, 809

## element tags (XML documents), 711

## Employee Records System program

- Database class, 34–38
- Employee class, 29–34
- user interface, 38–41

## empty element tag (XML documents), 711

## encrypting passwords, 502–504

## end() method, 565

## #endif preprocessor directive, 3

## endl manipulator, 383

## enumerated types, 10–11

## equal() algorithm, 97, 635–636

## equal\_range() algorithm, 96, 635

## errno macro, 403

## error checking

- containers, 563
- reusable code, 112

## error handling

- constructors, 427–428
- destructors, 428
- errno macro, 403
- integer function return codes, 403
- memory allocation errors, 424–426

## error logging, 528–530

## errors

- input streams, 387–388
- memory errors
  - accessing memory, 545–546
  - freeing memory, 544–545
- output streams, 382–383

## escape characters, 4

## evaluating reusable code, 81

## event handling

- chain of responsibility design pattern, 776–778
- observer design pattern, 778–781

## exceptions

- advantages of using, 403
- catching, 24–25, 402, 405–406, 732–733
- cleanup, 423
- containers, 563
- custom exceptions, 152
- defined, 23–24, 402

disadvantages of using, 404  
 effect on program efficiency, 471  
 file input/output, 404–405  
 hierarchy, 416–419  
 matching, 410–411  
 multiple exceptions, 408–410  
 objects, 406–408  
 polymorphism, 416  
 stack unwinding, 422–423  
 support, 89  
 throw lists, 412, 414–416  
 throwing, 24–25, 402, 405–406, 732–733  
 try-catch block, 24  
 uncaught exceptions, 411–412  
 unexpected exceptions, 413–414  
 writing exception classes, 419–421, 732  
 exceptions() **method**, 406  
**exclamation point (!) operator**, 8  
*Expert C Programming: Deep C Secrets*,  
 Peter Van Der Linden, 806  
**explicit conversions**, 211  
**explicit keyword**, 211  
**exposed interface**, 74  
**extending classes**, 224–225, 731–732  
**extending the STL (standard template library)**, 660  
**Extensible Markup Language (XML)**  
 attributes, 711  
 defined, 709–710  
 deserialization, 717  
 distributed objects, 723, 725–726  
 document structure, 710–711  
 Document Type Definition (DTD), 721–722  
 element tags, 711  
 empty element tag, 711  
 generating XML, 712–713  
 hierarchy, 710  
 learning curve, 709  
 namespaces, 711  
 output class, 713–717  
 parsing XML  
   DOM (Document Object Model) parser, 717  
   SAX (Simple API for XML) parser, 717  
   Xerces parser, 717–721  
 plain text format, 710  
 prolog, 710–711  
 root element, 711  
 serialization, 712, 723–726  
 SOAP (Simple Object Access Protocol), 726–728  
 text node, 711  
 validation, 721–723  
 XML Schema, 722–723  
**extern keyword**, 335–336  
**external linkage**, 333–334  
**extraction operator overloading**, 441–443  
**Extreme Programming (XP)**, 128–131  
*eXtreme Programming eXplained*, Kent Beck, 128, 808

## F

**facets**, 400  
**factory design pattern**, 760–766  
*Facts and Fallacies of Software Engineering*,  
 Robert L. Glass, 808  
 fail() **method**, 383  
**Farley, Jim**, *Java Distributed Computing*, 810  
**FIFO (first in, first out)**, 91  
**file streams**, 380, 393–394  
**files**  
   reading, 733  
   writing, 734  
**files streams**, 392  
 fill() **algorithm**, 98  
 fill\_n() **algorithm**, 98  
**FIFO (first-in, last-out)**, 92  
 find() **algorithm**, 96, 620–622, 633  
 find\_end() **algorithm**, 96  
 find\_first\_of() **algorithm**, 96, 633  
 find\_if() **algorithm**, 96, 622–623, 633  
**finding memory leaks**, 375–376  
**first in, first out (FIFO)**, 91  
**first-in, last-out (FILO)**, 92  
**fixed-format comments**, 142–143  
**fixing memory leaks**, 375–376  
**float variable type**, 7  
 flush() **method**, 382–383, 395  
**for loops**, 15  
 for\_each() **algorithm**, 98, 637–639  
**formatting code**, 152–154  
**forward iterators**, 564  
**Fowler, Martin**, *Refactoring: Improving the Design of Existing Code*, 808  
**Foxall, James**, *Practical Standards for Microsoft Visual Basic .NET*, 808  
**frameworks**  
   BeOS, 750  
   defined, 78, 750  
   Microsoft Foundation Classes (MFC), 750  
   model-view-controller (MVC), 750–752  
   omniORB, 702  
   reusable code, 78  
 free() **function**, 353  
**free software open-source movement**, 86  
**freeing memory**, 186  
**freeware**, 86  
**friend classes**, 208  
**friend methods**, 208  
 front() **method**, 568  
**fstream class**, 396  
**function call operator overloading**, 448–449  
**function object adapters**  
   binders, 627–628  
   negators, 628–629  
**function objects**  
   arithmetic, 624–625  
   calling member functions, 629–630

## function objects (continued)

- comparison, 625–626
- defined, 624
- logical, 627
- writing, 630–631

## function pointers, 372–373

## function templates, 295–299

## functional relationships, 69–70

## functions

- declaring, 16–17
- default parameters, 203–204
- defined, 16, 272
- `free()`, 353
- `getline()`, 387
- inline functions, 204–205, 472
- legacy C functions, 630
- `main()`, 3
- `malloc()`, 352–353
- `mem_fun_ref()`, 629–630
- parameters, 272
- `printf()`, 379
- `realloc()`, 355
- reusable code, 78
- `scanf()`, 379
- static variables, 336

## functors. See function objects

## Fundamental Law of Debugging, 527

## Futrell, Robert T., *Quality Software Project Management*, 808

# G

## Gamma, Erich, *Design Patterns: Elements of Reusable Object-Oriented Software*, 811

## garbage collection, 101–102, 370–371

## gcov code coverage tool, 511

## gdb (GNU Debugger), 556–558, 809

## General KeyServer Questions, Sassafras Software, 810

## general purpose interfaces, 116, 118

## `generate()` algorithm, 98

## `generate_n()` algorithm, 98

## generating XML, 712–713

## generic code. See code reuse; templates

## `get()` method, 385–386

## `getline()` function, 387

## `getline()` method, 387

## getters, 149

## Giru, Mike, *C++ FAQs (Second Edition)*, 804

## Glass, Robert L., *Facts and Fallacies of Software Engineering*, 808

## global functions and operator overloading, 433–434

## global scope, 343

## GNU Debugger (gdb), 556–558, 809

## GNU gprof Web site, 809

## GNU Operating System — Free Software Foundation Web site, 807

## `good()` method, 382–383

## gprof profiling tool, 479–483, 485–487

## grid computing, 694

# H

## half-open range, 565

## has-a relationship, 63, 66–69

## hash tables

- accessor operations, 690–691
- constructors, 687
- erase operations, 688–689
- insertion operations, 687–688
- methods, 672–675
- typedefs, 671–672
- writing, 662–670

## header files, 343–344

## heap

- algorithms, 645
- arrays, 354
- constructors, 167
- defined, 18
- multidimensional arrays, 359–360
- objects, 165
- sorting algorithms, 99

## Hello, World! program, 2

## Helm, Richard, *Design Patterns: Elements of Reusable Object-Oriented Software*, 811

## Hennessy, John L.

- Computer Architecture: A Quantitative Approach (Third Edition)*, 809

- Computer Organization & Design: The Hardware/Software Interface (Second Edition)*, 809

## Henning, Michi, *Advanced CORBA Programming with C++*, 810

## hex manipulator, 383, 389

## hiding methods, 264–265

## hierarchies

- defined, 70–71
- exceptions, 416–419
- iterators, 631
- parallel hierarchy, 253
- reusable code, 109–110

## How SETI@home Works, Ron Hipschman, 810

## Hughes, Cameron and Tracey

- Mastering the Standard C++ Classes: An Essential Reference*, 805

- Stream Manipulators and Iterators in C++*, 805

## Hungarian Notation, 150

# I

## IBM

- Rational Purify, 809
- Rational Quantify profiling tool, 479
- Rational Software, 809
- Rational Unified Process (RUP), 126–127, 808

## idea reuse, 50

## IDL (Interface Definition Language), 702–705

## `#ifdef [key]` preprocessor directive, 3

## `if/else` statements, 12

## `#ifndef [key]` preprocessor directive, 3

- `ifstream` class, **392**
- IIOIP (Internet Inter-ORB Protocol), 702**
- implementation**
  - abstract classes, 218–221
  - classes, 731
  - iterators, 565
  - mix-in classes, 749
  - subclasses, 732
  - versus interface, 73
- implicit conversions, 211**
- `#include [file]` **preprocessor directive, 3**
- `includes()` **algorithm, 99, 646**
- increment operators, 8**
  - overloading, 439, 441
  - overloading operators, 440
- indexing versus iterators, 575**
- inheritance**
  - casting
    - downcasting, 239–240
    - slicing, 239
    - upcasting, 239
  - child classes, 234
  - clients' view, 225–226
  - code reuse, 230
  - defined, 64–65
  - multiple inheritance, 72–73, 248–252
  - non-public inheritance, 269
  - overriding methods, 227–230
  - parent classes, 224–225, 234–237
  - polymorphism, 240–247
  - scope resolution operator (`::`), 237–239
  - subclasses
    - adding functionality, 231–233
    - copy constructors, 263–264
    - default arguments, 260–261
    - assignment (`=`) operator, 264
    - overridden methods, 230
    - replacing functionality, 233
    - view of inheritance, 226–227
  - superclasses, 224–225, 260–261
  - templates, 111–112, 295
  - weather prediction program, 230–233
- initializer lists, 171–172**
- initializing variables, 336–337**
- `initPointer()` **method, 737**
- inline functions, 204–205, 472**
- inline methods, 204–205, 472**
- in-memory streams, 390**
- `inner_product()` **algorithm, 97**
- `inplace_merge()` **algorithm, 99**
- input iterators, 564**
- input methods**
  - `get()`, 385–386
  - `getline()`, 387
  - `peek()`, 387
  - `printf()` function, 379
  - `putback()`, 387
  - `scanf()` function, 379
  - `seek()`, 392–393
  - `seekg()`, 393
  - `tell()`, 392–393
  - `tellg()`, 393
  - `unget()`, 386
- input streams**
  - bidirectional, 396–397
  - `cin`, 384–385
  - console, 380
  - defined, 3–4, 88, 379–380
  - errors, 387–388
  - facets, 400
  - files, 380, 392–394
  - `ifstream` class, 392
  - `istringstream` class, 391
  - linking, 395
  - locales, 398–400
  - manipulators, 388–389
  - objects, 389–390
  - source, 380
  - strings, 380, 391
  - wide characters, 397–398
  - `wifstream` class, 397
- `inRange()` **method, 185**
- insert iterators, 658–660**
- insertion operator overloading, 441–443**
- instances, 58–59**
- instantiating templates, 279–280**
- `int` **variable type, 7**
- integer function return codes, 403**
- integration tests, 523–525**
- Interface Definition Language (IDL), 702–705**
- interfaces**
  - abstract classes, 218–221
  - application programming interface (API), 74–75, 78
  - comments, 115–116
  - component interface, 75
  - customizability, 117
  - documentation, 115–116
  - exposed interface, 74
  - future use, 75
  - general purpose, 116, 118
  - intuitive, 113–114
  - operator overloading, 113–114
  - reusable code, 107–108, 112–118
  - subsystem interface, 75
  - versus implementation, 73
- internal linkage, 333–334**
- internationalization, 88, 397**
- Internet Inter-ORB Protocol (IIOIP), 702**
- Introduction to Algorithms (Second Edition)*, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, **807**
- intuitive interfaces, 113–114**
- invalid pointers, 377**
- I/O streams**
  - bidirectional, 396–397
  - console, 380

## I/O streams (continued)

- defined, 3–4, 88, 379–380
- facets, 400
- files, 380, 392–394
- in-memory, 390
- input streams, 384–389
- linking, 395
- locales, 398–400
- objects, 389–390
- output streams, 380–384
- `printf()` function, 379
- `scanf()` function, 379
- strings, 380, 390–392
- user, 380
- wide characters, 397–398

## is-a relationship, 64–69

`istream` class, 391

## iterator adapters

- defined, 656
- insert iterators, 658–660
- reverse iterators, 656–657
- stream iterators, 657–658

## iterator design pattern, 103

## iterator traits, 662

## iterators

- algorithms, 631–632
- bidirectional, 564
- `const_iterator`, 565, 574
- defined, 95
- dumb pointers, 565
- forward, 564
- half-open range, 565
- hierarchy, 631
- implementation, 565
- indexing, 575
- input, 564
- `iterator`, 565
- lists, 584
- maps, 599–600
- output, 564
- pointers, 565
- random access, 564
- safety, 574–575
- vectors, 572–577
- writing, 675–682

`iter_swap()` algorithm, 98

## J

**Java and C++ with JNI, 499–501**

**Java Cookbook, Ian F. Darwin, 806**

**Java Distributed Computing, Jim Farley, 810**

**JavaDoc tool, 142**

## job interview questions

- C++ basics, 783–784
- C++ quirks and oddities, 793–794
- classes, 789–792

- code reuse, 787
- coding style, 788–789
- cross-platform/cross-language applications, 798
- debugging skills, 799
- design patterns, 801
- distributed objects, 800–801
- efficient code, 797–798
- error handling, 796
- frameworks, 801
- inheritance, 792–793
- I/O streams, 795–796
- libraries, 786
- memory management, 794–795
- object design, 785–786
- objects, 789–790, 792
- overloading operators, 796–797
- patterns, 786
- program design, 784–785
- software engineering methods, 787–788
- Standard Template Library (STL), 799–800
- templates, 793
- testing skills, 798–799

**Johnson, Ralph, *Design Patterns: Elements of Reusable Object-Oriented Software*, 811**

**Josuttis, Nicolai M.**

*The C++ Standard Library: A Tutorial and Reference*, 805

*C++ Templates: The Complete Guide*, 806

## K

**Kernighan, Brian W, *The C Programming Language (Second Edition)*, 806**

**KeyServer (Sassafra Software), 694**

## keywords

- `asm`, 504
- `const`
  - constants, 25
  - methods, 333
  - pointers, 330–332, 362
  - proper use, 151
  - references, 26, 332
  - uses, 25
  - variables, 25, 330
- `delete`, 351
- `explicit`, 211
- `extern`, 335–336
- `new`, 351
- `public`, 269
- `static`, 333–335
- `template`, 276
- `typename`, 276
- `using`, 258–259
- `virtual`, 227–228

**Kirch-Prinz, Ulla, *C Pocket Reference*, 806**

**Knuth, Donald E.**

*The Art of Computer Programming Volume 1: Fundamental Algorithms (Third Edition)*, 807

*The Art of Computer Programming Volume 2: Seminumerical Algorithms (Third Edition)*, 807  
*The Art of Computer Programming Volume 3: Sorting and Searching (Third Edition)*, 807

**Kruchten, Philippe**, *Rational Unified Process: An Introduction (Second Edition)*, 808

**Kulchenko, Pavel**, *Programming Web Services with SOAP*, 810

## L

**Lajoie, Josée**, *C++ Primer (Third Edition)*, 803

language features, 472

language-level efficiency, 466–472

*Learning XML (Second Edition)*, T. Ray, 810

legacy C functions, 630

**Leiserson, Charles E.**, *Introduction to Algorithms (Second Edition)*, 807

lexicographical\_compare() algorithm, 97, 635–637

libraries

C standard library, 88

C++ standard library, 87, 89

defined, 78

open-source, 86–87

reusable code, 78

licensing

code reuse, 84

third-party applications, 85

life cycle

bugs, 508–509

objects, 165

life cycle models

defined, 120

Rational Unified Process (RUP), 126–127

Spiral Method, 123–126

Stagewise Model, 121–122

Waterfall Model, 122–123

line by line comments, 140–141

linkage

extern keyword, 335–336

external, 333–334

internal, 333–334

static keyword, 333–335

linking streams, 395

**Lippman, Stanley B.**, *C++ Primer (Third Edition)*, 803

**Lischner, Ray**, *C++ in a Nutshell*, 804

lists

accessing elements, 584

adding elements, 584–585

algorithms, 586–587

defined, 90, 565, 584

doubly linked lists, 90–91

iterators, 584

performance, 94

removing elements, 584–585

size, 585

splicing, 585–586

university or college enrollment example, 587–588

load balancer, 694

locales, 398–400

location transparency, 702

logger, 754–760

logging errors, 528–530

logical function objects, 627

**Lomow, Greg**, *C++ FAQs (Second Edition)*, 804

long variable type, 7

loops

defined, 14

do/while loops, 15

for loops, 15

while loops, 14–15

**Loudon, Kyle**, *Mastering Algorithms with C*, 807

lower\_bound() algorithm, 96, 635

## M

macros

assert macro, 540–541

errno macro, 403

preprocessor macros, 347

main() function, 3

make\_heap() algorithm, 99, 645

malloc() function, 352–353

manipulators

boolalpha, 383, 389

dec, 383, 389

endl, 383

hex, 383, 389

input streams, 388–389

noboolalpha, 383, 389

noshowpoint, 383

noskipws, 389

oct, 383, 389

output streams, 383–384

setfill, 383

setprecision, 383

setw, 383

showpoint, 383

skipws, 389

ws, 389

maps

bank account example, 602–604

defined, 93–94

iterators, 599–600

looking up elements, 600–602

operations, 596–599

removing elements, 602

marshalling (distributed objects), 696–700

*Mastering Algorithms with C*, Kyle Loudon, 807

*Mastering the Standard C++ Classes: An Essential Reference*, Cameron Hughes and Tracey Hughes, 805

matching exceptions, 410–411

mathematical utilities, 89

max() algorithm, 96, 632–633

max\_element() algorithm, 96, 633

## **Mayhew, David, *Efficient C++: Performance Programming Techniques*, 809**

### **members (classes)**

- access control, 159–160
- access specifiers, 159–160
- accessing, 161, 195–196
- const, 196–198
- const reference, 199
- declaring, 158
- pointers, 217–218
- reference, 198–199
- static, 195–196, 333
- static const, 197–198

### **members (references), 326**

`mem_fun_ref()` **function, 629–630**

### **memory**

- aliasing, 102
- dynamic memory
  - advantages of using, 350
  - allocation, 183–190, 351–352
  - arrays, 353–360
  - deallocation, 351–352
  - defined, 18–19
  - failure of memory allocation, 353
  - `free()` function, 353
  - `malloc()` function, 352–353
  - mental model, 350–351
  - orphaned memory, 351–352
  - `realloc()` function, 355
- freeing with destructors, 186
- heap
  - algorithms, 645
  - arrays, 354
  - constructors, 167
  - defined, 18
  - multidimensional arrays, 359–360
  - objects, 165
  - sorting algorithms, 99
- out-of-bounds memory, 378
- pointer arithmetic, 369–370
- pointers
  - arrays, 362–364
  - casting, 361–362
  - declaring, 19
  - dereferencing, 20–21
  - invalid pointers, 377
  - mental model, 360–361
  - smart pointers, 88–89, 101–102, 376–377, 424
  - variables, 20
- stack
  - arrays, 354
  - constructors, 166–167
  - defined, 18
  - error correlation, 594–595
  - multidimensional arrays, 357–358
  - objects, 164
  - operations, 594

- stack frames, 18–19
- underallocating strings, 374

**memory allocation operator overloading, 457–458**

**memory deallocation operator overloading, 457–458**

### **memory errors**

- accessing memory, 545–546
- allocation errors, 424–426
- debugging, 546–547
- freeing memory, 544–545

**memory leaks, 374–377, 544**

### **memory management**

- function pointers, 372–373
- garbage collection, 370–371
- object pools, 371

`merge()` **algorithm, 99, 644**

**metainformation comments, 139**

### **method behavior and run-time types**

- brute force approach, 742–743
- double dispatch, 741, 744–747
- single polymorphism with overloading, 743–744

**method overloading, 202–203**

**method templates, 285–290**

`methodPtr` **pointer, 217**

### **methods**

- access control, 159–160
- access specifiers, 159–160
- `add()`, 209
- `at()`, 568
- `back()`, 568
- `bad()`, 383
- `begin()`, 565
- calling, 161–163
- `clear()`, 383
- const, 200–202
- const keyword, 333
- constructor, 165
- `copyFrom()`, 193–194
- declaring, 158
- default parameters, 203–204
- defined, 26–29
- defining, 161
- destructor, 176–177
- `end()`, 565
- `exceptions()`, 406
- `fail()`, 383
- `flush()`, 382–383, 395
- friend, 208
- `front()`, 568
- `get()`, 385–386
- `getline()`, 387
- `good()`, 382–383
- hiding, 264–265
- `initPointer()`, 737
- inline methods, 204–205, 472
- `inRange()`, 185
- multi-methods, 741
- operator [ ], 446–447, 567, 599

- operator+, 210–212
  - operator overloading, 433–434
  - overriding
    - changing method characteristics, 253–256
    - clients' view, 229–230
    - private method, 259–260
    - protected method, 259–260
    - slicing, 230
    - special cases, 256–263
    - syntax, 228–229
    - virtual methods, 227–228
  - parameters, 254–256
  - peek(), 387
  - pointers, 217–218
  - pure virtual methods, 242–243
  - push\_back(), 569
  - put(), 381–382
  - putback(), 387
  - return types
    - changing, 253–254
    - covariant returns types, 253
  - scope resolution operator (::), 161
  - seek(), 392–393
  - seekg(), 393
  - seekp(), 393
  - size(), 569
  - static, 199–200, 333
  - str(), 391
  - tell(), 392–393
  - tellg(), 393
  - tellp(), 393
  - template classes, 277–279, 281–282
  - this pointer, 163–164
  - tie(), 395
  - unset(), 386
  - virtual methods, 227–228, 264–267
  - write(), 381–382
  - Meyers, Scott**
    - Effective C++ (Second Edition): 50 Specific Ways to Improve Your Programs and Designs*, 804
    - Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*, 805
    - More Effective C++: 35 New Ways to Improve Your Programs and Designs*, 804
  - Microsoft Foundation Classes (MFC), 750**
  - mImpl pointer, 220–221
  - min() algorithm, 96, 632–633
  - min\_element() algorithm, 96, 633
  - mismatch() algorithm, 97, 635–636
  - mix-in classes**
    - defined, 73
    - designing, 747–749
    - implementation, 749
    - uses, 749
    - XMLSerializable class, 723–724
  - mixing languages**
    - C and C++, 494–498
    - C++ with assembly code, 504–505
    - C++ with perl and shell scripts, 501–502
    - Java and C++ with JNI, 499–501
  - mod, equal (%) operator, 8**
  - mod (%) operator, 8**
  - model-view-controller (MVC), 750–752**
  - Modern C++ Design: Generic Programming and Design Patterns Applied*, Andrei Alexandrescu, 811
  - modifying algorithms, 98, 639**
  - modifying category of references, 324–325**
  - More Effective C++: 35 New Ways to Improve Your Programs and Designs*, Scott Meyers, 804
  - multidimensional arrays, 16, 357–360**
  - multimaps**
    - buddy lists, 605–608
    - defined, 93–94, 604–605
  - multi-methods, 741**
  - multiple dispatch, 741**
  - multiple exceptions, 408–410**
  - multiple inheritance, 71–73, 248–252**
  - multiplication, equal (\*=) operator, 8**
  - multiplication (\*) operator, 8**
  - multisets, 92–94, 610–611**
  - multithreaded programs, 547**
  - Muntz, Amy, Applied C++: Practical Techniques for Building Better Software*, 805
  - Musser, David R., STL Tutorial and Reference Guide (Second Edition)*, 805
  - MVC (model-view-controller), 750–752**
- ## N
- name ambiguity and multiple inheritance, 249–252**
  - nameserver, 706**
  - namespaces**
    - defined, 4–6
    - std namespace, 562
    - XML, 711
  - naming conventions, 148–151**
  - negators, 628–629**
  - nested classes, 206–207**
  - networked computing versus distributed computing, 695–696**
  - new command, 20**
  - new expression, 457–458**
  - new keyword, 351**
  - new line (\n) escape character, 4**
  - newsgroups, 804**
  - next\_permutation() algorithm, 98
  - noboolalpha manipulator, 383, 389
  - noncatastrophic bugs, 528
  - non-integral array indices, 447–448**
  - nonmodifying algorithms, 96, 633**
  - non-public inheritance, 269**
  - nonreproducible bugs, 543–544**
  - nonstandard strings, 23**
  - non-virtual destructors, 266–267**
  - non-Western character sets, 398**
  - noshowpoint manipulator, 383

`noskipws` manipulator, **389**

**not-a relationship**, **69**

`nth_element()` algorithm, **99**, **645**

**numerical processing algorithms**, **97**, **635**

## O

**Object Management Group's CORBA Web site**, **810**

### object pools

class template, 473–477

implementation of a pool class template, 478

memory management, 371

program efficiency, 473

### object relationships

defined, 63

functional, 69–70

has-a, 63, 66–69

is-a, 64–69

not-a, 69

**Object Request Broker (ORB) framework**, **706**

### object-oriented frameworks

BeOS, 750

defined, 750

Microsoft Foundation Classes (MFC), 750

model-view-controller (MVC), 750–752

**object-oriented programming (OOP)**, **26**, **58**

### objects

arrays, 355–356

assigning values to objects, 177–178

copy constructors, 172–174, 181, 190–191

creating, 164–166

destruction, 176–177

distributed objects

CORBA, 702–709

deserialization, 697

marshalling, 696–700

remote procedure call (RPC), 700–701

serialization, 696–700

unmarshalling, 697

XML, 712, 723, 725–726

dynamic memory allocation, 183–190

exceptions, 406–408

function objects

adapters, 627–629

arithmetic, 624–625

calling member functions, 629–630

comparison, 625–626

defined, 624

logical, 627

writing, 630–631

on the heap, 165

I/O streams, 389–390

life cycle, 165

out of scope, 176

overly general objects, 62

overobjectification, 61–62

passing by reference, 174–175

polymorphism, 66

return values, 180–181

on the stack, 164

temporary objects, 469–470

**observer design pattern**, **779–781**

`oct` manipulator, **383**, **389**

`ofstream` class, **392**

**omniORB framework**, **702**

**one-dimensional arrays**, **16**

**OOP (object-oriented programming)**, **26**, **58**

**Opdyke, William, *Refactoring: Improving the Design of Existing Code***, **808**

**Open Source Initiative**, **86**, **807**

**open-source libraries**, **86–87**

**open-source movements**, **86**

**operational algorithms**, **98**, **637**

### operator overloading

`add()` method, 209

arguments, 434

arithmetic operators, 212–215, 438–439

behavior, 435

binary logical operators, 441

bitwise operators, 441

built-in types, 216–217

comparison operators, 215–216

decrement operators, 439–441

dereferencing operators, 449–451

extraction operator, 441–443

function call operator, 448–449

global functions, 433–434

global `operator+` method, 211–212

implicit conversions, 211

increment operators, 439–441

insertion operator, 441–443

intuitive interfaces, 113–114

limitations, 432–433

memory allocation operators, 457–458

memory deallocation operators, 457–458

methods, 433–434

`operator*`, 451

`operator [ ]` method, 446–447, 567, 599

`operator delete`, 459–463

`operator+` method, 210–211

`operator new`, 459–463

operators to avoid, 435

rationale for, 432

return types, 434–435

subscripting operator, 443–446

summary table of overloadable operators, 435–438

### operators

ampersand (&), 8

ampersand, equal (&=), 8

assignment (=), 8, 178–180, 191–194, 264, 563

binary, 8

caret (^), 9

caret, equal (^=), 9

conditional, 13–14

- conversion operators, 453–455
- decrement (--), 8
- defined, 8
- division (/), 8
- division, equal (/=), 8
- exclamation point (!), 8
- increment (++), 8
- mod (%), 8
- multiplication (\*), 8
- multiplication, equal (\*=), 8
- plus (+), 8
- plus, equal (+=), 8
- precedence, 9–10
- scope resolution (::), 161, 237–239
- subtraction (-), 8
- subtraction, equal (-=), 8
- ternary, 8, 13
- typeid operator, 268–269
- unary, 8
- vertical line (|), 8
- optimization of return values, 470–471**
- ORB (Object Request Broker) framework, 706**
- order of declarations, 160**
- order of initialization of nonlocal variables, 336–337**
- orphaned memory, 351**
- `<ostream>` header file, 380
- `ostream` class, 391
- Oualline, Steve, *Practical C++ Programming (Second Edition)*, 803**
- out of scope objects, 176**
- out-of-bounds memory, 378**
- output iterators, 564**
- output methods**
  - `bad()`, 383
  - `clear()`, 383
  - `fail()`, 383
  - `flush()`, 382–383, 395
  - `good()`, 382–383
  - `printf()` function, 379
  - `put()`, 381–382
  - `scanf()` function, 379
  - `seek()`, 392–393
  - `seekp()`, 393
  - `tell()`, 392–393
  - `tellp()`, 393
  - `write()`, 381–382
- output streams**
  - bidirectional, 396–397
  - buffers, 382
  - console, 380
  - `cout`, 381
  - defined, 3–4, 88, 379–380
  - destination, 380
  - errors, 382–383
  - facets, 400
  - files, 380, 392–394
  - linking, 395
  - locales, 398–400
  - manipulators, 383–384
  - objects, 389–390
  - `ofstream` class, 392
  - `<ostream>` header file, 380
  - `ostream` class, 391
  - strings, 380, 391
  - wide characters, 397–398
  - `wofstream` class, 397
- overloading methods, 202–203**
- overloading operators**
  - `add()` method, 209
  - arguments, 434
  - arithmetic operators, 212–215, 438–439
  - behavior, 435
  - binary logical operators, 441
  - bitwise operators, 441
  - built-in types, 216–217
  - comparison operators, 215–216
  - decrement operators, 439–441
  - dereferencing operators, 449–451
  - extraction operator, 441–443
  - function call operator, 448–449
  - global functions, 433–434
  - global `operator+` method, 211–212
  - implicit conversions, 211
  - increment operators, 439–441
  - insertion operator, 441–443
  - intuitive interfaces, 113–114
  - limitations, 432–433
  - memory allocation operators, 457–458
  - memory deallocation operators, 457–458
  - methods, 433–434
  - `operator*`, 451
  - `operator [ ]` method, 446–447, 567, 599
  - `operator delete`, 459–463
  - `operator+` method, 210–211
  - `operator new`, 459–463
  - operators to avoid, 435
  - rationale for, 432
  - return types, 434–435
  - subscripting operator, 443–446
  - summary table of overloadable operators, 435–438
- overloading template functions, 297–298**
- overly general objects, 62**
- overobjectification, 61–62**
- overriding methods**
  - changing method characteristics, 253–256
  - clients' view, 229–230
  - `private` method, 259–260
  - `protected` method, 259–260
  - slicing, 230
  - special cases, 256–263
  - syntax, 228–229
  - virtual methods, 227–228

## P

`pair` class, 595–596

parallel hierarchy, 253

### parameters

- default parameters, 203–204
- functions, 272
- methods, 254–256
- references, 326
- templates
  - nontype, 283–285, 305–306
  - syntax, 282–283
  - template, 303
  - type, 299–304
  - zero-initialization syntax, 307

### parent classes

- constructors, 234–235
- destructors, 235–237
- inheritance, 224–225

parentheses ( ) in code, 154

### parsing XML

- DOM (Document Object Model) parser, 717
- SAX (Simple API for XML) parser, 717
- Xerces parser, 717–721

`partial_sort()` algorithm, 99, 645

`partial_sort_copy()` algorithm, 99

`partial_sum()` algorithm, 97

`partition()` algorithm, 99, 645

### passing by reference

- defined, 21
- language-level efficiency, 467–469
- objects, 174–175
- references, 327

### passing by value

- defined, 21
- disallowing, 194
- references, 327

password encryption, 502–504

patterns. *See* design patterns

### Patterson, David A.

- Computer Architecture: A Quantitative Approach (Third Edition)*, 809
- Computer Organization & Design: The Hardware/Software Interface (Second Edition)*, 809

`peek()` method, 387

### performance

- big-O notation, 82–83
- bitsets, 94
- defined, 465
- deque, 94
- documentation, 83
- lists, 94
- maps, 94
- multimaps, 94
- multisets, 94
- priority queues, 94
- queues, 94
- reusable code, 81
- sets, 94

- stacks, 94
- vectors, 94

perl, 501–502

pipe (|) operator, 8

platforms. *See* cross-platform applications

plus (+) operator, 8

plus, equal (+=) operator, 8

plus, plus (++) operator, 8

POA (Portable Object Adapter), 704

pointer arithmetic, 369–370

### pointers

- arrays, 362–364
- casting, 361–362
- const keyword, 330–332, 362
- containers, 563
- data members (classes), 217–218
- declaring, 19
- dereferencing, 20–21
- invalid pointers, 377
- iterators, 565
- mental model, 360–361
- `methodPtr`, 217
- methods, 217–218
- `mImpl`, 220–221
- references, 151–152, 325–330
- smart pointers
  - `auto_ptr` template, 88–89
  - containers, 563
  - memory leaks, 376–377, 424
  - reference counting, 736–737
  - `SuperSmartPointer` implementation, 737–741
  - uses, 101–102
- `this`, 163–164
- variables, 20

### polymorphism

- code reuse, 66
- defined, 66
- double dispatch, 741, 744–747
- exceptions, 416
- inheritance, 240–247
- single polymorphism with overloading, 743–744

### pools

- object pools
  - class template, 473–478
  - memory management, 371
  - program efficiency, 473
  - thread pools, 479

`pop_heap()` algorithm, 99, 645

Portable Object Adapter (POA), 704

*The Portland Pattern Repository*, Cunningham and Cunningham, 811

*Practical C++ Programming (Second Edition)*, Steve Oualline, 803

*Practical Standards for Microsoft Visual Basic .NET*, James Foxall, 808

`#pragma` preprocessor directive, 3

Prata, Stephen, *C++ Primer Plus*, 804

precedence of operators, 9–10

prefix comments, 141–142

- prefixes, 149**
  - preprocessor directives, 2–3**
  - preprocessor macros, 347**
  - `prev_permutation()` **algorithm, 98**
  - `printf()` **function, 379**
  - Prinz, Peter, C Pocket Reference, 806**
  - priority queues**
    - defined, 91–92, 94
    - error correlation, 592–594
    - operations, 591–592
  - `private` **access control, 159–160**
  - `private` **assignment operators, 194**
  - `private` **copy constructors, 194**
  - Problem Solving with C++: The Object of Programming (Fourth Edition)*, Walter Savitch, 803**
  - procedures, 57–58. See functions**
  - profiling programs, 479**
  - profiling tools**
    - `gprof`, 479–487
    - Rational Quantify, 479
  - program design**
    - abstraction, 47–48
    - C++ features and challenges, 46–47
    - Chess program example, 51–55
    - defined, 44
    - importance of, 44–46
    - procedures, 57–58
    - prototypes, 85
    - reusable code, 49–50
  - Programming Abstractions in C: A Second Course in Computer Science*, Eric S. Roberts, 806**
  - Programming Web Services with SOAP*, James Snell, Doug Tidwell, and Pavel Kulchenko, 810**
  - programs**
    - auditing voter registrations, 648–653
    - Chess, 51–55
    - efficiency
      - defined, 465–466
      - design-level, 466, 472–479
      - exceptions, 471
      - inline functions, 472
      - inline methods, 472
      - language-level, 466–472
      - RTTI, 471–472
      - virtual methods, 471
    - Employee Records System
      - Database class, 34–38
      - Employee class, 29–34
      - user interface, 38–41
    - Hello, World!, 2
    - performance, 465
    - profiling, 479
  - prolog (XML documents), 710–711**
  - properties, 59**
  - `protected` **access control, 159**
  - prototypes, 85**
  - proxy design pattern, 766–768**
  - `public` **access control, 159**
  - `public` **keyword, 269**
  - pure virtual methods, 242–243**
  - `push_back()` **method, 569**
  - `push_heap()` **algorithm, 99, 645**
  - `put()` **method, 381–382**
  - `putback()` **method, 387**
- ## Q
- Quality Software Project Management*, Robert T. Futrell, Donald F. Shafer, and Linda Isabell Shafer, 808**
  - quality-assurance, 507**
  - queues**
    - defined, 91, 588
    - first in, first out (FIFO), 91
    - network buffer packet example, 589–591
    - operations, 588–589
    - performance, 94
  - quotation mark ( \ " ) escape character, 4**
- ## R
- random access iterators, 564**
  - `random_shuffle()` **algorithm, 99, 645**
  - Rational Purify (IBM), 809**
  - Rational Quantify profiling tool, 479**
  - Rational Software (IBM), 809**
  - Rational Unified Process: An Introduction (Second Edition)*, Philippe Kruchten, 808**
  - Rational Unified Process (RUP), 126–127, 808**
  - raw output methods, 381–382**
  - Ray, T., *Learning XML (Second Edition)*, 810**
  - reading from a file, 733**
  - read-only access, 446–447**
  - `realloc()` **function, 355**
  - refactoring, 147**
  - Refactoring: Improving the Design of Existing Code*, Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts, 808**
  - reference counting, 736–737**
  - reference data members (classes), 198–199**
  - references**
    - `const` keyword, 26, 199, 332
    - data members, 326
    - defined, 23, 323
    - modifying, 324–325
    - parameters, 326
    - pass-by-reference, 327
    - pass-by-value, 327
    - pointers, 151–152, 325–330
    - return values, 327
    - variables, 23, 323–324
    - versus pointers, 151–152
  - regression testing, 525**
  - `reinterpret_cast`, **340, 342**

## relationships (objects)

- defined, 63
- functional, 69–70
- has-a, 63, 66–69
- is-a, 64–69
- not-a, 69

**reliability of distributed computing, 694**

**remote procedure call (RPC)**

- SOAP (Simple Object Access Protocol), 726–728
- stub method, 700
- writing, 700–701

`remove()` **algorithm, 98, 642**

`remove_copy()` **algorithm, 98, 642**

`remove_copy_if()` **algorithm, 98, 642**

`remove_if()` **algorithm, 98, 642**

**rendering farms, 694**

`replace()` **algorithm, 98, 641**

`replace_copy()` **algorithm, 98**

`replace_copy_if()` **algorithm, 98**

`replace_if()` **algorithm, 98, 641**

**reproducible bugs, 541–543**

**return types**

- changing, 253–254
- covariant returns types, 253
- operator overloading, 434–435

**return values**

- objects, 180–181
- optimization, 470–471
- references, 327

**reusable code**

- abstraction, 107–108
- advantages, 79
- aggregation, 110
- categories of, 78
- disadvantages, 79–80
- error checking, 112
- evaluating, 81
- frameworks, 78
- goals, 106–107
- hierarchies, 109–110
- inheritance, 230
- interfaces, 107–108, 112–118
- libraries, 78
- licensing, 84
- limitations, 81
- open-source libraries, 86–87
- performance, 81
- philosophy, 106
- platforms, 83–84
- polymorphism, 66
- program design, 49–50
- safeguards, 112
- stand-alone functions or classes, 78
- subsystems, 109
- support, 84–85
- templates, 110–112
- tips for writing, 49–50, 108–109

- reusable ideas, 50**
- `reverse()` **algorithm, 98, 643**
- reverse iterators, 656–657**
- `reverse_copy()` **algorithm, 98, 643**
- reversible containers, 682**
- ring buffers, 535–540**
- Ritchie, Dennis M., *The C Programming Language (Second Edition)*, 806**
- Rivest, Ronald L., *Introduction to Algorithms (Second Edition)*, 807**
- Roberts, Don, *Refactoring: Improving the Design of Existing Code*, 808**
- Roberts, Eric S.**
  - The Art and Science of C: A Library Based Introduction to Computer Science*, 806
  - Programming Abstractions in C: A Second Course in Computer Science*, 806
- Romanik, Philip, *Applied C++: Practical Techniques for Building Better Software*, 805**
- root cause of bugs, 528**
- root element (XML documents), 711**
- `rotate()` **algorithm, 98**
- `rotate_copy()` **algorithm, 98**
- RPC (remote procedure call)**
  - SOAP (Simple Object Access Protocol), 726–728
  - stub method, 700
  - writing, 700–701
- RTTI (Runtime Type Identification)**
  - `dynamic_cast`, 267–268
  - effect on program efficiency, 471–472
  - `typeid` operator, 268–269
- running unit tests, 515**
- run-time debug mode, 535**
- Runtime Type Identification (RTTI). See RTTI (Runtime Type Identification)**
- RUP (Rational Unified Process), 126–127, 808**

## S

- safeguards for reusable code, 112**
- Saini, Atul, *STL Tutorial and Reference Guide (Second Edition)*, 805**
- Sassafras Software**
  - General KeyServer Questions*, 810
  - KeyServer, 694
- Savitch, Walter, *Problem Solving with C++: The Object of Programming (Fourth Edition)*, 803**
- SAX (Simple API for XML) parser, 717**
- scalability of distributed computing, 693–694**
- `scanf()` **function, 379**
- scope resolution (defined), 343**
- scope resolution operator (::), 161, 237–239**
- scripting, 502**
- `search()` **algorithm, 96, 633**
- Search for Extra-Terrestrial Intelligence (SETI), 694**
- `search_n()` **algorithm, 96**
- `seek()` **method, 392–393**

- `seekg()` **method**, 393
- `seekp()` **method**, 393
- selective instantiation**, 281
- self-documenting code**, 144–145
- sequential containers**, 565, 691
- serialization**
  - C++, 696–700
  - XML, 712, 723–726
- `set_difference()` **algorithm**, 99, 646
- `setfill` **manipulator**, 383
- SETI (Search for Extra-Terrestrial Intelligence)**, 694
- SETI@home project**, 694
- `set_intersection()` **algorithm**, 99, 646
- `setprecision` **manipulator**, 383
- sets**
  - access control list example, 609–610
  - defined, 92–94, 608
- `set_symmetric_difference()` **algorithm**, 99, 646
- setters**, 149
- `set_union()` **algorithm**, 99, 646
- `setw` **manipulator**, 383
- Shafer, Donald F. and Linda Isabell, *Quality Software Project Management***, 808
- shareware**, 86
- shell scripts**, 501–502
- short variable type**, 7
- short-circuit logic**, 14
- `showpoint` **manipulator**, 383
- Simonyi, Charles, inventor of Hungarian Notation**, 150
- Simple API for XML (SAX) parser**, 717
- Simple Object Access Protocol (SOAP)**, 726–728
- singleton design pattern**, 754–760
- `size()` **method**, 569
- skeletons**, 704
- `skipws` **manipulator**, 389
- slicing**, 230, 239
- smart constants**, 150
- smart pointers**
  - `auto_ptr` template, 88–89
  - containers, 563
  - memory leaks, 376–377, 424
  - reference counting, 736–737
  - `SuperSmartPointer` implementation, 737–741
  - uses, 101–102
- smoke testing**, 525
- Snell, James, *Programming Web Services with SOAP***, 810
- SOAP (Simple Object Access Protocol)**, 726–728
- software bugs**
  - buffer overflow errors, 378
  - Bugzilla bug-tracking tool, 509–510
  - catastrophic bugs, 528
  - error logging, 528–530
  - Fundamental Law of Debugging, 527
  - life cycle, 508–509
  - noncatastrophic bugs, 528
  - nonreproducible, 543–544
  - regression testing, 525
  - reproducible, 541–543
  - root cause, 528
  - tips for avoiding bugs, 528
- software design**
  - abstraction, 47–48
  - C++ features and challenges, 46–47
  - Chess program example, 51–55
  - defined, 44
  - importance of, 44–46
  - procedures, 57–58
  - prototypes, 85
  - reusable code, 49–50
- software engineering**
  - bugs, 120
  - business risks, 120
  - developing your own processes and methodologies, 132–133
  - development time, 120
  - Extreme Programming (XP), 128–131
  - failures, 119–120
  - feature creep, 120
  - software triage, 132
  - unpredictability, 120
- software life cycle models**
  - defined, 120
  - Rational Unified Process (RUP), 126–127
  - Spiral Method, 123–126
  - Stagewise Model, 121–122
  - Waterfall Model, 122–123
- software triage**, 132
- `sort()` **algorithm**, 99, 643–644
- `sort_heap()` **algorithm**, 99, 645
- sorting algorithms**, 99
- sourceforge.net Web site**, 807
- spaces in code**, 154
- specializing templates**
  - full class specialization, 290–293, 295
  - partial class specialization, 307–313
- Spinellis, Diomidis, *Code Reading: The Open Source Perspective***, 808
- Spiral Method**, 123–126
- A Spiral Model of Software Development and Enhancement***, Barry W. Boehm, 807
- splicing lists**, 585–586
- Spolsky, Joel, *The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)***, 805
- spreadsheet example**, 157–158
- `stable_partition()` **algorithm**, 99
- `stable_sort()` **algorithm**, 99, 643
- stack**
  - arrays, 354
  - constructors, 166–167
  - defined, 18
  - error correlation, 594–595
  - multidimensional arrays, 357–358
  - objects, 164
  - operations, 594
- stack frames**, 18–19

**stack unwinding, 422–423**

**stacks (STL containers), 92, 94**

**Stagewise Model, 121–122**

**standard template library (STL). See STL (standard template library)**

**standard template library (STL) algorithms. See STL (standard template library) algorithms**

**standard template library (STL) containers. See STL (standard template library) containers**

**start-time debug mode, 532–535**

**static const data members (classes), 197–198**

**static data members (classes), 195–196, 333**

**static keyword, 333–335**

**static methods, 199–200**

**static variables, 336**

**static\_cast, 339–340, 342**

**std namespace, 562**

**Stein, Clifford, *Introduction to Algorithms (Second Edition)*, 807**

**STL (standard template library)**

defined, 89

extending, 660

omissions, 100

std namespace, 562

unsupported functionality, 100

**STL (standard template library) algorithms**

`accumulate()`, 97, 623–624

`adjacent_difference()`, 97

`adjacent_find()`, 96, 633

`binary_search()`, 96, 635

choosing, 100

comparison, 97, 635

`copy()`, 98, 640–641

`copy_backward()`, 98

`count()`, 97, 635

`count_if()`, 97, 635

defined, 95, 620

`equal()`, 97, 635–636

`equal_range()`, 96, 635

`fill()`, 98

`fill_n()`, 98

`find()`, 96, 620–622, 633

`find_end()`, 96

`find_first_of()`, 96, 633

`find_if()`, 96, 622–623, 633

`for_each()`, 98, 637–639

`generate()`, 98

`generate_n()`, 98

`includes()`, 99, 646

`inner_product()`, 97

`inplace_merge()`, 99

iterator traits, 662

iterators, 95, 631–632

`iter_swap()`, 98

`lexicographical_compare()`, 97, 635–637

lists, 586–587

`lower_bound()`, 96, 635

`make_heap()`, 99, 645

`max()`, 96, 632–633

`max_element()`, 96, 633

`merge()`, 99, 644

`min()`, 96, 632–633

`min_element()`, 96, 633

`mismatch()`, 97, 635–636

modifying, 98, 639

`next_permutation()`, 98

nonmodifying, 96, 633

`nth_element()`, 99, 645

numerical processing, 97, 635

operational, 98, 637

`partial_sort()`, 99, 645

`partial_sort_copy()`, 99

`partial_sum()`, 97

`partition()`, 99, 645

`pop_heap()`, 99, 645

`prev_permutation()`, 98

`push_heap()`, 99, 645

`random_shuffle()`, 99, 645

`remove()`, 98, 642

`remove_copy()`, 98, 642

`remove_copy_if()`, 98, 642

`remove_if()`, 98, 642

`replace()`, 98, 641

`replace_copy()`, 98

`replace_copy_if()`, 98

`replace_if()`, 98, 641

`reverse()`, 98, 643

`reverse_copy()`, 98, 643

`rotate()`, 98

`rotate_copy()`, 98

`search()`, 96, 633

`search_n()`, 96

`set_difference()`, 99, 646

`set_intersection()`, 99, 646

`set_symmetric_difference()`, 99, 646

`set_union()`, 99, 646

`sort()`, 99, 643–644

`sort_heap()`, 99, 645

sorting, 99

`stable_partition()`, 99

`stable_sort()`, 99, 643

`swap()`, 96, 632

`swap_ranges()`, 98

`transform()`, 98, 639–640

`unique()`, 98, 643

`unique_copy()`, 98

`upper_bound()`, 96, 635

utility, 96, 632–633

writing, 660–662

**STL (standard template library) containers**

accessing fields of object elements, 573–574

arrays, 611–612

assignment operator, 563

associative, 595, 683–687

bitsets, 93

copy constructor, 563

- defined, 89–90
  - deque, 91, 565, 584
  - destructors, 563
  - element requirements, 562–563
  - error checking, 563
  - exceptions, 563
  - hash tables
    - accessor operations, 690–691
    - constructors, 687
    - erase operations, 688–689
    - insertion operations, 687–688
    - methods, 672–675
    - typedefs, 671–672
    - writing, 662–670
  - iterators, 95, 564–565
  - lists, 90–91, 565, 584
  - maps, 93, 596–599
  - multimaps, 93, 604–605
  - multisets, 92–93, 610–611
  - performance comparison, 94
  - pointers, 563
  - priority queues, 91–92, 591–592
  - queues, 91, 588–589
  - reference semantics, 562
  - reversible, 682
  - sequential, 691
  - sequential containers, 565
  - sets, 92–93, 608
  - stacks, 92
  - std namespace, 562
  - streams, 613–618
  - strings, 612–613
  - value semantics, 562
  - vectors, 90, 565–566
  - writing, 662–670
- STL Tutorial and Reference Guide (Second Edition),**  
**David R. Musser, Gillmer J. Derge, and Atul Saini, 805**
- str() method, 391**
- Stream Manipulators and Iterators in C++, Cameron Hughes**  
**and Tracey Hughes, 805**
- streams**
- bidirectional, 396–397
  - console, 380
  - containers, 613–618
  - defined, 3–4, 88, 379–380
  - facets, 400
  - files, 380, 392–394
  - in-memory, 390
  - input streams, 384–389
  - linking, 395
  - locales, 398–400
  - objects, 389–390
  - output streams, 380–384
  - strings, 380, 390–392
  - user, 380
  - wide characters, 397–398
- stress testing, 526**
  - string streams, 380, 390–392**
  - strings**
    - C++, 21–22
    - containers, 612–613
    - C-style, 21, 365–366
    - nonstandard, 23
    - string class, 88, 367–369
    - string literals, 366–367
    - underallocating, 374
  - stringstream class, 397**
  - Stroustrup, Bjarne, *The C++ Programming Language*, 1, 804**
  - structs, 11–12**
  - stub method, 700**
  - stubs, 703–704**
  - stylistic consistency (when writing code), 155**
  - subclasses, 234**
    - assignment (=) operator, 264
    - copy constructors, 263–264
    - creating, 731–732
    - default arguments, 260–261
    - implementation, 732
    - inheritance
      - adding functionality, 231–233
      - overridden methods, 230
      - replacing functionality, 233
      - view of, 226–227
    - template classes, 293–294
  - subscripting operator overloading, 443–446**
  - subsystems**
    - interfaces, 75
    - reusable code, 109
  - subtraction, equal (==) operator, 8**
  - subtraction (-) operator, 8**
  - superclasses**
    - default arguments, 260–261
    - defined, 65
    - inheritance, 224–225
  - SuperSmartPointer implementation, 737–741**
  - support for reusable code, 84–85**
  - Sutter, Herb, *Sutter's Mill: Befriending Templates*, 806**
  - swap() algorithm, 96, 632**
  - swap\_ranges() algorithm, 98**
  - switch statements, 12–13**
  - symbolic debugger, 542**
  - system tests, 525**
- T**
- tab (t) escape character, 4**
  - tabs, 154**
  - tell() method, 392–393**
  - tellg() method, 393**
  - tellp() method, 393**
  - template classes**
    - method definitions, 277–279, 281–282
    - subclasses, 293–294

## template classes (continued)

- syntax, 276–277
- uses, 273
- writing, 273–275, 734–736

**template keyword, 276**

**templates**

- `auto_ptr` smart pointer, 88–89
- code bloat, 111, 281
- compiler processes, 280–281
- compiler support, 272
- function templates, 295–299
- inheritance, 111–112, 295
- instantiating, 279–280
- method templates, 285–290
- parameterization, 272, 276–277
- parameters
  - nontype, 283–285, 305–306
  - syntax, 282–284
  - template, 303
  - type, 299–304
  - zero-initialization syntax, 307
- recursion, 314–322
- reusable code, 110–112
- selective instantiation, 281
- specialization
  - full class specialization, 290–293, 295
  - partial class specialization, 307–313
- type requirements, 281
- type specification, 279–280
- type-safe, 111

**temporary objects, 469–470**

**ternary operator, 8, 13**

**testing**

- black box testing, 507
- bugs
  - bug-tracking tools, 509–510
  - life cycle, 508–509
- comments, 526
- integration tests, 523–525
- quality-assurance (QA), 507
- regression testing, 525
- responsibilities of team members, 508
- smoke testing, 525
- stress testing, 526
- system tests, 525
- tips for successful testing, 526
- unit tests
  - code coverage, 511
  - cppunit open-source framework, 516–523
  - defined, 510–511
  - example unit tests, 515–516
  - granularity, 512–513
  - process, 512–514
  - running, 515
  - sample data and results, 514
  - writing, 511, 514–515
- white box testing, 507

- text node (XML documents), 711**
- Thinking in C++, Volume 1: Introduction to Standard C++ (Second Edition)*, Bruce Eckel, 803
- Thinking in C++, Volume 2: Practical Programming (Second Edition)*, Bruce Eckel and Chuck Allison, 804
- third-party applications, 85**
- this pointer, 163–164**
- thread pools, 479**
- throw lists (exceptions), 412, 414–416**
- throwing exceptions, 402, 405–406, 732–733**
- Tidwell, Doug, *Programming Web Services with SOAP*, 810**
- tidy open-source command-line program, 713**
- tie() method, 395**
- transform() algorithm, 98, 639–640**
- try-catch block, 24**
- type specification templates, 279–280**
- typeid operator, 268–269**
- typename keyword, 276**
- types**
  - casts, 338–342
  - operator overloading, 216–217
  - typedefs, 337–338
  - wchar\_t, 397
- type-safe templates, 111**

## U

- unary operators, 8**
- uncaught exceptions, 411–412**
- underallocating strings, 374**
- unexpected exceptions, 413–414**
- unset() method, 386**
- Unicode, 398**
- Unicode Web site, 805**
- unique() algorithm, 98, 643**
- unique\_copy() algorithm, 98**
- unit tests**
  - code coverage, 511
  - cppunit open-source framework, 516–523
  - defined, 510–511
  - example unit tests, 515–516
  - granularity, 512–513
  - process, 512–514
  - running unit tests, 515
  - sample data and results, 514
  - writing unit tests, 511, 514–515
- unmarshalling (distributed objects), 697**
- unsigned int variable type, 7**
- unsigned long variable type, 7**
- unsigned short variable type, 7**
- upcasting, 239**
- upper\_bound() algorithm, 96, 635**
- user streams, 380**
- using keyword, 258–259**
- utility algorithms, 96, 632–633**

**V**

**valgrind memory leak checking program, 377**  
**Valgrind memory-debugging tool, 809**  
**validation of XML, 721–723**  
**value assignment to objects, 177–178**  
**Van Der Linden, Peter, *Expert C Programming: Deep C Secrets*, 806**  
**van der Vlist, Eric, *XML Schema*, 810**  
**van Heesch, Dimitri, *Doxygen*, 808**  
**Vandevoorde, David, *C++ Templates: The Complete Guide*, 806**  
**variable-length argument lists, 345–347**  
**variables**  
  bool, 7  
  casting, 7–8  
  char, 7  
  coercing, 7  
  const keyword, 25, 330  
  converting types, 7  
  declaring, 6  
  defined, 6  
  double, 7  
  enumerated types, 10–11  
  float, 7  
  initializing, 336–337  
  int, 7  
  long, 7  
  order of initialization of nonlocal variables, 336–337  
  passing by reference, 21  
  passing by value, 21  
  pointers, 20  
  references, 23, 323–324  
  short, 7  
  static, 336  
  structs, 11–12  
  unsigned int, 7  
  unsigned long, 7  
  unsigned short, 7  
**variable-sized arrays, 355**  
**vectors**  
  allocator type parameter, 566  
  assignment operator, 571  
  at() method, 568  
  back() method, 568  
  bool specialization, 583–584  
  comparing, 571–572  
  constructors, 569–570  
  copying, 571  
  defined, 90, 565  
  destructors, 569–570  
  dynamic-length, 568–569  
  element type parameter, 566  
  fixed-length arrays, 566–567  
  front() method, 568  
  initial element value, 567–568  
  iterators, 572–577  
  memory allocation scheme, 577–578

operator [ ] method, 567  
  performance, 94  
  push\_back() method, 569  
  round-robin scheduling, 578–583  
  size and capacity, 578  
  size() method, 569  
  template parameter, 566  
  vector header file, 566  
**vertical line (|) operator, 8**  
**Vinoski, Steve, *Advanced CORBA Programming with C++*, 810**  
**virtual base classes, 269–270**  
**virtual destructors, 236–237**  
**virtual keyword, 227–228**  
**virtual methods, 227–228, 264–267, 471**  
**Vlissides, John, *Design Patterns: Elements of Reusable Object-Oriented Software*, 811**  
**voter registration auditing example program, 648–653**

**W**

**Waterfall Model, 122–123**  
**wchar\_t type, 397**  
**weather prediction program, 230–233**  
**Web Services Essentials, Ethan Cerami, 810**  
**Web sites**  
  Altova Software xmlspy, 810  
  C++ Resources Network, 805  
  cppunit open-source unit testing framework, 516  
  GNU gprof, 809  
  GNU Operating System—Free Software Foundation, 807  
  Object Management Group's CORBA, 810  
  Open Source Initiative, 807  
  sourceforge.net, 807  
  Unicode, 805  
**while loops, 14–15**  
**white box testing, 507**  
**wide characters, 397–398**  
**wifstream class, 397**  
**wofstream class, 397**  
**write() method, 381–382**  
**writing algorithms, 660–662**  
**writing classes**  
  access specifiers, 159–160  
  class definitions, 158  
  code example, 730–731  
  exception classes, 419–421, 732  
  extending existing classes, 731–732  
  implementation, 731  
  members (defined), 158  
  methods (defined), 158  
  order of declarations, 160  
  template classes, 273–275, 734–736  
**writing code**  
  const keyword, 151  
  constants, 151  
  custom exceptions, 152  
  decomposition, 145–147

### writing code (continued)

- documentation, 136–145
- formatting, 152–154
- importance of “good” code, 135–136
- naming conventions, 148–151
- references versus pointers, 151–152
- stylistic consistency, 155

**writing constructors, 166**

**writing containers, 662–670**

**writing conversion operators, 453–455**

**writing function objects, 630–631**

**writing iterators, 675–682**

**writing remote procedure call (RPC), 700–701**

**writing to a file, 734**

**writing unit tests, 511, 514–515**

ws **manipulator, 389**

## X

**Xerces parser, 717–721**

### XML (Extensible Markup Language)

- defined, 709–710
- deserialization, 717
- distributed objects, 723, 725–726
- document structure, 710–711
- Document Type Definition (DTD), 721–722
- element tags, 711
- empty element tag, 711
- generating XML, 712–713

- hierarchy, 710
- learning curve, 709
- namespaces, 711
- output class, 713–717
- parsing XML
  - DOM (Document Object Model) parser, 717
  - SAX (Simple API for XML) parser, 717
  - Xerces parser, 717–721
- plain text format, 710
- prolog, 710–711
- root element, 711
- serialization, 712, 723–726
- SOAP (Simple Object Access Protocol), 726–728
- text node, 711
- validation, 721–723
- XML Schema, 722–723

**XML Schema, Eric van der Vlist, 810**

**XMLSerializable class, 723–724**

**xmlspy (Altova Software), 723, 810**

**XP (Extreme Programming), 128–131**

## Y

**Yourdon, Edward, *Death March*, 132, 808**

## Z

**zero-initialization syntax, 289**