

# Index

## SYMBOLS

- +, addition operator, 69–72
- \a, alert escape sequence, 53–54
- &, ampersand symbol, 260–261, 263, 268, 270–271, 511, 519–520
- address-of operator, 511
- call-by-reference parameters, 260–261, 263, 268, 270–271, 519–520
- memory locations and, pointers, 511, 519–520
- >, arrow operator, 742, 744
- =, assignment operator, 69, 74, 81–82, 493, 511–512, 514, 569, 757–758, 860–861
- arithmetic operators and, 69, 74
- dynamic data structures and, 757–758
- inheritance and, 860–861
- objects used with, 569
- overloading, 680–682
- pointers and, 511–512, 514, 757–758
- variables and, 69, 511–512
- vectors, 493
- \*, asterisk symbol, 22, 509–512, 961, 964–965
- dereferencing operator, 510–511, 961, 964–965
- multiplication operator, 22, 70
- pointer variable declaration, 509–512
- \, backslash, 23
- \, backslash escape sequence, 53
- &&, Boolean and operator, 78–79, 112–116
- !, Boolean not operator, 79, 113, 118
- ||, Boolean or operator, 78, 80, 112–116
- { }, braces, 24–25, 82, 84–86, 121–123, 137–138, 543, 547
- C++ programming layout, 24–25
- conditional statements and, 82
- local variable declaration, 137–138
- loop body execution, 84–86
- nested statements and, 121–123, 137–138
- structure member names, 543, 547
- :, colon symbol, 557–558, 600–601
- derived class separation, 600–601
- inheritance and, 600–601
- scope resolution operator, 557–558
- ,, comma for separation in declarations, 44, 431
- //, comment symbols, 93–94,
- ==, comparison equal to operator, 78, 81–82, 456–457, 487
- >, comparison greater than operator, 78
- >=, comparison greater than or equal to operator, 78, 80
- <, comparison less than operator, 78
- <=, comparison less than or equal to operator, 78
- !=, comparison not equal to operator, 78–79
- , decrement operators, 87–91, 143–144, 960–961, 967–969
- <> >>, direction arrows, 21
- #, directive notation, 25
- /, division operator, 70–72
- ., dot (calling) operator, 313, 545, 550
- \", double quote escape sequence, 53
- " ", double quotes for string characters, 64–65
- =, equal sign, 22, 456–457
- >>, extraction operator, 309, 316–318, 464, 650–658
- n!, factorial function, 230–231
- ++, increment operators, 87–91, 141–144, 960–961, 967–969, 971
- <<, insertion operator, 310, 316–318, 329, 464, 650–658
- \n, new line instruction, 23, 338–360, 345–346
- \n, new-line instruction, 53, 58
- \0, null character, 453–454, 456
- ( ), parentheses, 71, 78–79, 84–86, 130, 184, 191, 197
- arguments, 184

- arithmetic order, 71
- Boolean expressions, 78–79, 84–86
- controlling expressions, 130
- predefined functions and, 184, 191, 197
- return statements, 197
- type casting and, 191
- < >, predefined function header files, 184–186
- \\, real backslash escape sequence, 53
- %, remainder operator, 70–71
- ;, semicolons, 24, 44, 131, 146, 149–150, 547
- end of declarations, 24, 44, 131
- for* statements, 146, 149–150
- structure definitions, 547
- ' ', single quotes for constant characters, 65
- [ ], square brackets, 378–379, 392–393, 427, 431, 489, 492
- arrays using, 378–379, 392–393
- multidimensional arrays and, 427, 431
- variable declaration and, 378–379, 489
- vectors using, 489, 492
- +, string concatenation, 66–67.
- , subtraction operator, 70
- \t, tab escape sequence, 53
- \_, underscore symbol for identifiers, 70, 475–477
- <cstring> library, 458–459
- <string> library, 472, 474
- A**
- abs function, 186–187
- Absolute value functions, 186–187
- Abstract data types (ADT), 588–597, 705–715
- application file, 714
- case study: *DigitalTime*—a Class Compiled Separately, 706–712
- class types for, 588–597
- compiling programs using, 705–715
- implementation files, 592–597, 708–715
- information hiding, 597
- interface files, 592–593, 705–707
- private member function changes, 593, 597, 705–716
- programming, 588–597
- reusable components of, 715
- separate compilation using, 705–715
- writing, rules for, 591–592
- Accessor functions, 567–568, 626
- Adapter container classes, 979–983
- addition (+), 70
- Addresses, 4–5, 32, 509–511, 528–529
- address-of operator (&), 511
- arithmetic performed on pointers, 528–529
- memory location and, 4–5, 32
- pointers, 509–511, 528–529
- ADT, *see* Abstract data types (ADT)
- Algorithms, 12–16, 30–32, 212–213, 278, 400–402, 792–793, 811–812, 926–938, 991–1004
- abstraction, 926–938
- array programs, 400–402
- bubble sort, 421–423
- design, 212–213, 278, 400–402, 792–793
- development of, 12–14
- generic, 991–1004
- implementation phase, 15
- logic errors, 30–31
- object-oriented programming (OOP) for, 16–17
- problem solutions using, 12–15
- problem-solving phase, 15
- procedural abstraction and, 212–213, 278
- program design using, 15–16
- programming use of, 12–16, 32
- pseudocode, 213
- recursion programs, 792–793, 811–812
- templates for, 926–938, 991–1004
- Ampersand symbol (&), 260–261, 263, 268, 270–271, 511, 519
- Ancestor class, 844
- And operator (&&), 78–79, 112–116
- Appending to a file, 320–322
- Application file, ADT, 714
- Arguments, 183–184, 197–198, 200–201, 265–268, 332, 346–349, 389–398, 459–460, 548–549, 585–586, 747
- array parameters for, 391–396
- arrays and, 389–398, 459–460
- C strings, 459–460
- call-by-reference parameters, 265–268
- call-by-value parameters, 197–198
- character I/O, 346–349
- const* parameter modifier for, 394–397
- constructors without, 585–586
- default, 348–349
- formal parameters and, 197–198, 200–201
- function calls using, 183–184, 197–198, 265–268
- function subtasks using, 265–268
- functions in arrays, 389–398
- incorrect order of, 200–201

- Arguments (*continued*)
  - indexed variables as, 389–391
  - linked lists as, 747
  - parentheses ( ) and, 184
  - predefined functions, 183–184, 459–460
  - programmer-defined functions, 197–198, 200–201
  - streams as, 332, 346–347
  - structures as, 548–549
- Arithmetic functions, 1019
- Arithmetic operators, 69–74, 112–116, 528–529
  - addition (+), 70
  - assignment operator (=) and, 69, 74
  - Boolean operations compared to, 112–115
  - data types and, 69–72
  - division (/), 70–72
  - double* used with, 70
  - expressions and, 69–72
  - int* used with, 70–72
  - multiplication (\*), 70
  - negative integers in, 71
  - Op* shorthand notation, 74
  - parentheses ( ) for order of, 71
  - pointer addresses using, 528–529
  - remainder (%), division with, 70–71
  - subtraction (-), 70
  - variables and, 69–72
- Array parameter, 391–396
- Array variables, 521–523
- Arrays, 377–450, 453–472, 521–527, 530–532, 660–682, 1027–1028
  - arguments to functions, 389–398
  - base type, 379
  - C strings, 453–472
  - case study: Production Graph, 398–409
  - class members, 664–667
  - classes and, 660–682, 942–943
  - const* modifier, 376–382, 394–397
  - constructor calls for, 661
  - declaring, 378–384, 426–427, 453–454
  - dynamic, 521–527, 530–532, 667–682
  - errors in, 383–384
  - for* loops used with, 380
  - functions and, 389–398
  - index (subscript), 379, 383–384, 1027–1028
  - indexed variables and, 379–386, 389–391, 426, 431
  - initializing, 386, 454–455
  - int* data types, 378–382
  - memory locations, 382–383, 393–394
  - multidimensional, 425–431, 530–532
  - overloading, 1027–1028
  - parameters, 391–397, 414, 426–427
  - partially filled, 411–413
  - programming with, 411–423
  - referencing, 378–384
  - searching, 414–416
  - size of, 379–382, 394, 411–414, 426–427
  - sorting, 417–423
  - square brackets [ ] used for, 378–379, 392–393, 427
  - strings as types of, 453–472
  - subtasks for functions of, 399–400
  - two-dimensional, 427, 531–532
  - variables in, 378–386, 389–391, 453–460
- Arrow (->) operator, 742, 744
- Ask-before-iterating looping technique, 157, 159
- Assembly language, 8
- assert* macro, 290–291
- Assignment operator (=), 69, 74, 81–82, 493, 511–512, 514, 569, 680–682, 757–758, 860–861
- Assignment statements, 45–49, 511–512
  - pointers and, 511–512
  - variable values and, 45–49
- Associative containers, 983–990
- Asterisk symbol (\*), 22, 70, 509–512
- atof* function, 467
- atoi* function, 467–468
- atoll* function, 467
- Augusta, Ada, 12–13
- auto
  - C++11, 63
  - using with containers, 990
  - variable declaration using, 964
- Automatic variables, 518
- B**
- Babbage, Charles, 12–13
- Backslash \ use, 23
- Base (stopping) cases, 798
- Base class, 834, 836–837, 848–850, 858
- Base type, 379
- Bidirectional iterators, 966–969
- Big-O notation, 994–995
- Binary digits, 4
- Binary tree, 761–762
- Bits (binary digits), 4, 32
- Black box analogy, 204–207.
  - See also* Procedural abstraction
- Blocks, 135–137, 226–227
  - branching statements as, 137
  - functions and, 226–227
  - local variables and, 135–137, 226–227
  - nested, 137
  - scope, 137, 226–227
  - statement, 137
- bool* values, 66, 116, 199
  - data type, 66

- int*, converting to, 116–118
- programmer-defined functions returning, 199
- Boolean expressions, 66, 77–79, 84–87, 112–119
- and (&&) operator used in, 78–79, 112–116
- arithmetic operations
  - compared to, 112–115
- branching mechanisms
  - using, 77–79, 112–119
- complete evaluation, 116
- data values, 66
- evaluating, 112–116
- int* value conversion, 116–118
- looping mechanisms using, 84–87, 112–119
- not (!) operator, 79, 113, 118
- or (||) operator, 78, 80, 112–116
- parentheses ( ) for, 78–79, 84–86
- precedence rules, 114–115
- short-circuit evaluation, 115
- subexpressions, 115
- true/false* values, 66, 116
- truth tables, 112–114
- Braces { }, 24–25, 82, 84–86, 121–123, 137–138, 543, 547
- Branching mechanisms, 75–82, 112–139, 203
  - and operator (&&) for, 78–79
- blocks, 135–137
- Boolean expressions, 77–79, 112–120
- braces { } used for, 82, 121–123, 137–138
- break* statements, 131–133
- C++ flow of control using, 75–82
- comparison operators for, 77–82
- compound statements, 82
- controlling expression, 130–132
- flow of control using, 75–82, 112–139
- if-else* statements, 75–82, 120–128
- indenting, 120–121, 123–125
- local variables, 135–137
- menus, 133–134
- multiway, 120–139
- nested statements, 120–123, 137
- or operator (||) for, 78, 80
- programmer-defined function calls in, 203
- string of inequalities from, 80–81
- switch* statements, 128–135
- break* statements, 131–133, 153–154
- branching mechanisms, 131–133
- flow of control using, 131–133, 153–154
- loop mechanisms, 153–154
- looping mechanisms, 153–154
- nested loops using, 154
- switch* statements, 131–133
- Bubble sort, 421–423
- Bug, 29. *See also* Debugging
- Bytes, 4–5, 32
- C**
- C++ programming, 39–110
  - arithmetic operators, 69–74
  - assignment statements, 45–49
  - asterisk symbol (\*), 22
  - backslash (\) use, 23
  - braces { }, 24–25, 82, 84–86
  - branching mechanisms, 75–82
  - cin* (input) statements, 21–23, 56–57
  - comments, 93–95
  - compilers and, 24–25
  - compiling, 26–28
  - compound statements, 82
  - constants, 95–97
  - cout* (output) statements, 21–23, 50–52
  - data types, 44–45, 60–74
  - debugging, 29–31
  - declaration of variables, 21–23, 44–45
  - direction arrows (<> >>) , 21
  - directives #, 25, 52–53
  - expressions, 69–74
  - flow of control, 74–92
  - increment and decrement operators, 87–91
  - indentation, 93
  - input, 21–23, 56–59
  - input/output (I/O), 50–59
  - instructions, 19–23
  - language, 18–19
  - line breaks, 24
  - loop mechanisms, 84–91, 98
  - main()* function, 25
  - names and, 49, 95–97
  - object code, 26–27
  - output, 21–23, 50–56
  - programmer role, 19
  - return* statement, 25–26
  - running, 26–28
  - spacing, 24, 26
  - statements, 21–26, 40–50, 82
  - user role, 19
  - variables, 21–23, 40–50, 60–69
- C++11 programming, 27, 63–64
  - auto, 63
  - constructor delegation in, 587–588
  - conversion between strings and numbers, 488
  - data values, 63–64
  - decltype*, 64
  - member initialization in, 587
  - nullptr* in, 745
  - range-based, 386–387

- C strings, 453–472, 484–488
  - <cstring> library, 458–459
  - arguments, 459–460
  - arrays, 453–472
  - declaration of, 453–454
  - equality operators = and ==
    - used for, 456–457
  - extraction (>>) operator used for, 464
  - functions, 457–460, 467–468
  - getline function, 465–466
  - initializing, 454–455
  - input/output (I/O), 464–466
  - insertion (<<) operator used for, 464
  - null (\0) character and, 453–454, 456
  - number conversions, 466–470
  - parameters, 460
  - predefined functions, 457–460, 467–468
  - robust input, 468, 470–471
  - strcat function, 459–460
  - strcmp function, 457–460
  - strcpy function, 457–460
  - string object conversion, 487–488
  - values, 456–458
  - variables, 453–460
- Call-by-reference parameters, 259–266, 519–520
  - ampersand symbol (&)
    - for, 260–261, 263, 268, 270–271, 519–520
  - arguments, 265–268
  - call-by-value combined with, 268–271
  - function calls, 259–260
  - memory locations and, 260–261
  - pointers, 519–520
- Call-by-value parameters, 197–198, 224–226, 268–271, 674–675
  - arguments for, 197–198
  - call-by-reference combined with, 268–271
- classes and, 674–675
  - dynamic arrays and, 674–675
  - local variables as, 224–226, 270–271
- Calls (invocations), 183–188, 196–198, 208–211, 259–260, 265–268, 583–584
  - absolute value functions, 186–187
  - arguments and, 183–184, 197–198, 265–268
  - call-by-reference parameters, 259–260
  - call-by-value parameters, 197–198
  - constructors, 583–584
  - functions, 183–188, 196–198, 208–211, 259–260, 265–268
  - header files (< >) and, 184–186
  - #include directives, 184–186
  - loop body as, 208
  - nested loops and, 208–211
  - predefined functions, 183–188
  - procedural abstraction and, 208–211
  - programmer-defined functions, 196–198
  - return statements and, 196–197
- capacity( ) function, 493–494
- catch block, 900–901, 901–902, 908–909
- catch-block parameter, 900–902
- Central processing unit (CPU), 3, 6–7
- char data type, 64–66
- Characters, 64–65, 68, 338–360, 1022
  - blank spaces and, 338–339
  - data values, 64–65
  - default arguments, 348–349
  - editing text files, 355–357
  - eof function for, 353–354
  - functions, 348–349, 1022
  - get function for, 338–341
  - input/output (I/O), 338–349
  - isspace function, 358–359
  - member functions, 338–354
  - new-line (/n), 338–360, 345–346
  - new\_line( ) function for, 343–345, 347–348
  - predefined functions, 356, 358–360
  - put function for, 341–342
  - putback function for, 342–343
  - stream parameters and, 346–347
  - toupper and tolower
    - functions for, 358–360
  - values returned, 358–360
  - whitespace, 68, 358
- Child class, 600, 834, 844
- Chips, computer processors and, 6
- cin (input) statements, 21–23, 56–57
- Classes, 17, 66–68, 312–315, 472–488, 541–617, 619–682, 762–765, 833–892, 904–905, 973–990
  - abstract data types (ADT), 588–597
  - adapter, 979–983
  - ancestor, 844
  - arrays and, 660–682
  - base, 834, 836–837, 848–850, 858
  - C++ programming and, 17
  - call-by-value parameters and, 674–675
  - child, 600, 834, 844
  - constructors for, 576–588, 661, 668–671
  - containers, 973–990
  - copy constructors for, 675–679

- defining, 600–603
- derived, 598–603, 834–836, 837–845, 854–856, 860–861
- destructors for, 671–673
- dot operator (.) for, 557–558
- dynamic arrays and, 667–682
- encapsulation, 556
- exceptions, 904–905
- file I/O and, 312–315
- friend functions, 620–642
- hierarchies, 599–600
- inheritance and, 598–603, 833–892
- linked lists of, 762–765
- member functions of, 312–314, 554–558, 570–574, 576–588
- member variables, 664–667
- object-oriented programming (OOP) and, 17
- objects and, 312–315, 554, 566, 569, 576–588
- overloading operators, 643–660
- parent, 600–601, 834, 844
- private members used in, 559–568
- public members used in, 559–568
- redefining functions, 853–856
- scope resolution operator (::) for, 557–558
- streams and, 312–315
- string, 66–68, 472–488
- stringvar, 668–671
- structures compared to, 542–550, 575
- templates for, 973–990
- close function, 310–311, 318
- Coding, 213–214, 278–280, 400–407, 793–794, 812–816
  - array programs, 400–407
  - procedural abstraction and, 213–214, 278–280
  - recursion programs, 793–794, 812–816
- Colon (:), 557–558, 600–601
- Comma (,) separation in declarations, 44
- Comments, C++ programming and, 93–95
- Compact discs (CDs), 6
- Comparison operators, 77–82, 482, 487
  - and operator (&&) for, 78–79
  - equal to (==), 78, 81–82
  - greater than (>), 78
  - greater than or equal to (>=), 78, 80
  - less than (<), 78
  - less than or equal to (<=), 78
  - not equal to (!=), 78–79
  - or operator (||) for, 78, 80
  - string class and, 482, 487
  - string of inequalities from, 80–81
- Compiler programs, 9–11, 24–32, 704–718
  - abstract data types (ADT) interface for, 705–715
  - C++ programming, 24–28, 32
  - compiling process, 26–28
  - error messages, 30–31
  - #ifndef directive, 25, 26, 716–718
  - #include directive, 25, 26, 716
  - language translation using, 9–11
  - line breaks, 24
  - linking code, 9–11
  - object code, 9–11, 26–27
  - separate compilation, 704–718
  - spacing, 24, 26
  - syntax error, 30
  - testing, 27–29
- Complete evaluation, 116
- Compound statements, 82
- Computer systems, 2–12
  - compilers, 9–11
  - hardware, 2–7
  - input/output devices, 3
  - languages for, 8–11
  - linkers, 9–11
  - mainframe, 2
  - memory, 3–6
  - network, 2
  - operating systems, 7
  - personal (PC), 2
  - processor (CPU), 3, 6–7
  - programs, 2, 7–11
  - software, 2, 7–8
- Concatenation (+), strings, 66–67
- const modifier, 96–97, 376–382, 394–397, 638–642
  - array declaration using, 376–382
  - array parameters, 394–397
  - C++ programming using, 96–97
  - friend functions and, 638–642
  - inconsistent use of, 397
- Constant array parameters, 395
- Constant iterators, 970–971
- Constant parameters, 638–642
- Constants, 60–62, 65, 95–97, 119–120, 221–223, 473, 636
  - data types, 60–62
  - declared, 96
  - enumerated types, 119–120
  - friend functions and, 636
  - functions and, 221–223
  - global named, 221–223
  - naming, 95–97
  - numbers, leading zeros in, 636
  - single quotes (') for characters, 65
  - string class conversion, 473



- Constructors, 473–474, 492, 576–588, 661, 668–671, 675–679, 845–848, 860–861
    - arrays and, 661, 668–671, 675–679
    - calling (invoking), 583–584, 661
    - classes and, 576–588, 668–671, 675–679
    - copy, 675–679, 860–861
    - default, 473–474, 584–585, 661
    - dynamic arrays, 668–671, 675–679
    - inheritance and, 845–848, 860–861
    - initialization of objects
      - using, 576–583
    - member functions as, 576–588
    - no arguments and, 585–586
    - overloaded, 578
    - size of arrays and, 668–671
    - string class and, 473–474
    - vectors and, 492
  - Container modifying algorithms, 1001–1002
  - Containers, 973–990, 995–996
    - access running times, 995–996
    - adapter classes, 979–983
    - associative, 983–990
    - auto, using with, 990
    - deque, 976
    - doubly linked lists, 974
    - efficiency of, 990–991
    - initializing, 990
    - map class, 983–990
    - priority\_queue class, 979–983
    - queue class, 979–983
    - ranged for, using with, 990
    - sequential, 974–979
    - set class, 983–990
    - singly linked lists, 974
    - stack class, 979–983
    - templates for, 973–990
    - type definitions in, 979
  - Controlling expression, 130–132
  - Copy constructors, 675–679, 860–861
  - Count-controlled loops, 158
  - cout (output) statements, 21–23, 50–52, 289–290
    - debugging with, 289–290
    - direction arrows (<> >>), 21
      - program output using, 21–23, 50–52
    - streams, as
    - variable declaration and, 21–23
- D**
- Dangling pointers, 517, 522–523
  - Data, computer programs and, 7–8
  - Data abstractions, templates for, 939–948
  - Data types, 44–45, 60–74, 95–97, 119–120, 942
    - arithmetic operators and, 69–74
    - bool*, 66
    - Boolean, 66
    - C++11, 63–64
    - char*, 64–66
    - character, 64–65
    - compatibility of, 68–69
    - constants as, 60–62, 65, 95–97, 119–120
    - double*, 44, 60–64
    - enumerated, 119–120
    - expressions and, 69–74
    - float*, 63
    - floating-point notation, 61–63
    - int*, 44, 60–62, 63, 70–72
    - integer, 60–62
    - long*, 62–63
    - names for declaration, 44–45
    - numeric, 44, 60–64
    - Op* shorthand notation, 74
    - short*, 63
    - string class and, 66–68
    - templates for, 942
    - variables as, 44–45, 60–74
  - Debugging, 29–31, 162–164, 281–287, 287–291
    - assert macro for, 290–291
    - bugs, 29
    - code, 290
    - common errors, 287
    - cout statement for, 289–290
    - error messages, 30–31
    - functions, 281–287, 287–291
    - localizing errors, 288–290
    - logic errors, 30–31
    - loops, 162–164
    - off-by-one error, 162
    - retesting changes, 164
    - run-time errors, 30
    - second opinions and, 287
    - syntax errors, 30
    - testing programs for, 29–31, 281–287
    - tracing variables, 162–163, 288
    - warning messages, 30
  - Decimal (.) notation, 55–56, 61
  - Declaration, 21–23, 44–45, 48–49, 193, 195–196, 199–201, 275–281, 308–309, 378–384, 426–427, 453–454, 489–490
    - arrays, 378–384, 426–427
    - cin (input) statements for, 21–23
    - comma (,) for separation in, 44, 431
    - const modifier and, 376–382
    - cout (output) statements
      - for, 21–23
    - C-string variables, 453–454
    - double* variable type, 44

- functions, 193, 195–196, 199–201, 275–281
  - illegal ranges, 383–384
  - indexed variables, 379–384
  - initializing in, 48–49
  - int* variable type, 21, 44, 378–380
  - memory and, 382–383
  - multidimensional arrays, 426–427
  - postconditions, 275–281
  - preconditions, 275–281
  - programmer-defined functions, 193, 195–196, 199–200
  - semicolon (;) for end of, 44
  - square brackets [ ] used for, 378–379, 489
  - streams, 308–309
  - type names and, 44–45
  - variables, 21–23, 44–45, 48–49, 64, 378–382, 453–454, 489–490
  - vectors, 489–490
  - Declared size, 379
  - decltype*, 64
  - Decrement operators (--), 87–91, 143–144, 967–969
  - Default arguments, 348–349
  - Default constructors, 473–474, 584–585, 661
  - delete* operator, 517–518, 524–527, 530–531
  - Deque, 976
  - Dereferencing (\*) operator, 510–511, 961, 964–965
  - Derived classes, 598–603, 834–836, 837–845, 854–856, 860–861, 913
    - assignment (=) operators used for, 860–861
    - colon (:) for separation of, 600–601
    - constructors used in, 845–848
    - copy constructors used in, 860–861
    - defining, 600–603
    - destructors used in, 861
    - exception specification in, 913
    - implementation of, 834–836
    - inheritance and, 598–603, 834–836, 837–845, 854–856, 860–861
    - redefining functions, 853–856
  - Descendants, 844
  - Destructors, 671–673, 861, 875–876
    - dynamic arrays, 671–673
    - inheritance and, 860–861
    - polymorphism and, 875–876
    - virtual, 875–876
  - Digital video discs (DVDs), 6
  - digit\_to\_int* function implementation, 635–636
  - Direction arrows (<> >>), 21
  - Directives (#), 25, 52–53
  - Diskettes (floppy disks), 6
  - Division operator (/), 70–72
  - do-while* loop statements, 87–91, 139–140, 154
    - break* statement for, 154
    - execution of, 87, 139–140
    - infinite, 87–91
    - syntax of, 87–88, 139–140
  - Dot (.) operator, 313, 545, 550
  - double*, 44, 55–56, 60–64, 70
    - arithmetic operators and, 70
    - decimal (.) notation for, 61
    - exponent (e) notation for, 61
    - floating-point notation of, 61–62
    - numeric data type, 44, 60–64
    - output values from, 55–56
    - scientific notation of, 61–62
    - variable type, 44
  - Double quotes (" ") for string characters, 64–65
  - Double-precision numbers, 60–61
  - Doubly linked lists, 760–761, 974
  - Drivers, function testing using, 282–284
  - Dynamic arrays, 521–527, 530–532, 667–682, 740, 757–758
    - array variables and, 521–523
    - assignment operator (=) and, 757–758
    - call-by-value parameters and, 674–675
    - classes and, 667–682
    - constructors for, 668–671
    - copy constructors for, 675–679
    - creating and using, 522–527
    - delete* operator, 524–527, 530–531
    - destructors for, 671–673
    - linked lists and, 740, 757–758
    - multidimensional, 530–532
    - new* operator, 524–527
    - pointer arithmetic and, 528–529
    - pointer variables and, 521–523, 527, 740, 757–758
    - size of, 668–671
    - square brackets [ ] used for, 524–527, 526–527
    - stringvar* class, 668–671
    - variables, 521–523, 527, 740, 757–758
- ## E
- Echoing input, 58
  - Empty statements, 150
  - Encapsulation, 17
  - #endif* directive, 716–717
  - endl* instruction, 54–55
  - eof* function, 353–354
  - equal* function, 620–626
  - Equal to comparison operator (==), 78, 81–82, 456–457



- Errors, 29–31, 287–290, 316, 383–384, 431, 638–642, 874–875
    - arrays and, 383–384, 431
    - bugs, 29
    - commas between index variables, 431
    - common, 287
    - compiler, 30–31, 875
    - constant parameters for, 638–639
    - debugging, 287–290
    - file I/O, 316
    - index variables out of range, 383
    - localizing, 288–290
    - logic, 30–31
    - messages, 30–31, 316
    - polymorphism and, 874–875
    - run-time, 31
    - syntax, 30
    - testing for, 30–31
    - tracing variables, 162–163, 288
    - virtual member functions and, 874–875
    - warning messages compared to, 30
  - Escape sequences, 53–55
  - Exceptions, 893–924
    - catch*-block parameter, 900–902
    - catch* block used for, 900–901, 901–902, 908–909
    - class hierarchies, 917
    - classes defined for, 904–905
    - derived classes and, 913
    - functions, throwing in, 909–911
    - handler, 900
    - handling, 893–924
    - memory, testing for, 917–918
    - multiple, 904, 906–909
    - nested try-catch blocks, 916
    - overuse of, 916–917
    - programming techniques for, 914–918
    - rethrowing, 918
    - specification, 911–913
    - throw list, 911–913
    - throw* statement used for, 898–900, 909–911
    - throwing exceptions, 909–911, 914–916
    - trivial, 909
    - try* block used for, 898–899, 901
    - try-throw-catch* mechanism in, 898, 901–903
    - uncaught, 916
  - Executable statements. *See* Statements
  - Executing programs, 8
  - exit* function, 315, 318
  - Exit-on-flag loop termination, 159
  - Exponent (e) notation, 61
  - Expressions, 69–74. *See also*
    - Arithmetic operators;
    - Boolean expressions
  - External file name, 310
  - Extraction operator (>>), 309, 316–318, 464, 650–658
- ## F
- fabs* function, 187
  - factorial* (n!) function, 230–231
  - fail* function, 314
  - Files, 6, 306–323, 332–337, 353–358, 588–597, 705–715
    - abstract data types (ADT), 588–597, 705–715
    - appending, 320–322
    - application, 714
    - character I/O and, 353–358
    - close* function used for, 310–311, 318
    - computer memory and, 6
    - end of, 332–335, 353–354
    - eof* function used for, 353–354
    - error messages, 316
    - exit* function used for, 315, 318
    - external name, 310
    - extraction operator (>>) for, 309, 316–318
    - fail* function used for, 314
    - implementation, 592–597, 708–715
    - include* directives used for, 309, 318, 329
    - input/output (I/O), 306–323, 332–337
    - insertion operator (<<) for, 316–318, 329
    - interface, 592–593, 705–707, 713
    - member functions, 312–314
    - memory storage and, 6
    - names and, 308–310, 318
    - namespaces and, 335–336
    - open* function used for, 309–310, 318
    - opening successfully, 309–310, 315
    - permanent storage, as, 307–308
    - reading, 308
    - separate compilation of, 705–715
    - streams and, 306–338
    - text editing, 355–357
    - writing, 308–310
  - First-in/first-out (FIFO) data structure, 771
  - Fixed-point notation, 326
  - Flags, 159, 325–327
  - Flash drives, 6
  - float* data type, 63
  - Floating-point notation, 61–63
  - Flow of control, 74–92, 111–164
    - Boolean expressions for, 77–79, 112–120
    - branching mechanisms, 75–82, 112–139

- C++ programming and, 74–92
- comparison operators for, 77–82
- compound statements, 82
- enumerated types, 119–120
- increment and decrement operators, 87–91, 141–144
- loop mechanisms, 84–91, 98, 112–120, 139–155
- for* statements, 144–150, 154, 380
- arrays using, 380
- empty (null) statements, 150
- multistatement body, 148–149
- numeric calculations using, 143–146
- semicolons (;) and, 146, 149–150
- variables and, 145–146
- Formal parameters. *See* Parameters
- Forward iterators, 969
- Freestore, 516–517
- friend functions, 620–642
  - accessor functions and, 626
  - const parameter modifier, 638–642
  - constant parameters, 638–639
  - `digit_to_int` implementation, 635–636
  - equal, 620–626
  - leading zeros in number constants, 636
  - Money class, example for, 628–635
  - nonmember functions, as, 624–628
  - private members, access to, 624
  - syntax, 627
- Function body, 196
- Function declaration, 193, 195–196, 199–201
- Function definition, 193–194, 196–197, 201–203, 791, 798, 937–938
- Function headers, 196, 200
- Functions, 181–250, 251–303, 312–314, 323–338, 338–349, 389–398, 414, 457–460, 467–468, 567–568, 620–642, 791–807, 850, 853–856, 864–876, 909–911, 1019–1026. *See also* Calls (invocations)
  - arguments and, 183–184, 197–198, 265–268, 389–394, 414
  - arithmetic, 1019
  - array size and, 414
  - arrays as arguments, 391–394, 414
  - arrays in, 389–398
  - C++ library, 1019–1025
  - C string, 457–460, 467–468
  - call-by-reference, 259–266
  - call-by-value parameters, 197–198, 268–271
  - calls (invocations), 183–188, 196–198, 208–211, 259–260, 265–268
  - case study: Production Graph, 398–409
  - character, 348–349, 1022
  - `const` parameter modifier, 394–397, 638–642
  - debugging, 281–287, 287–291
  - declaration, 193, 195–196, 199–201, 275–281
  - default arguments, 348–349
  - definition, 193–194, 196–197, 201–203, 791, 798
  - `digit_to_int` implementation, 635–636
  - driver programs for, 282–284
  - equal, 620–626
  - factorial ( $n!$ ), 230–231
  - flags and, 325–327
  - formatting output using, 323–338
  - friend, 620–642
  - graph, 407
  - indexed variables as arguments, 389–391
  - inheritance and, 850, 853–856
  - inline, 1026
  - input/output (I/O), 323–349, 338–349, 1020–1021
  - local variables and, 218–229, 270–271
  - manipulators, 329
  - member, 312–314, 338–349
  - member functions accessor, 551–552
  - mutator, 567–568
  - names, 221–224, 232–238
  - nonmember, 624–628
  - not inherited, 850, 859–860
  - overloading names, 232–238
  - overriding, 869
  - parameters, 197–201, 207–208, 224–226, 259–266, 391–397
  - polymorphism and, 864–876
  - predefined, 183–192, 457–460, 467–468
  - procedural abstraction and, 204–217, 273–281
  - programmer-defined, 193–203
  - random number generator, 188–189, 1024
  - recursive, 791–807
  - redefining functions, 853–856
  - return* statements, 196–197, 202, 255–259
  - returning an array, 397–398
  - scale*, 402–407
  - signature, 857
  - stream I/O, 323–338
  - string, 1023

- Functions (*continued*)
  - stub, 284–286
  - subtasks, 251–303, 399–400
  - tasks, recursion for, 791–803
  - testing, 214–217, 281–287
  - throwing exceptions in, 909–911
  - top-down design for, 182–183, 398–409
  - trigonometric, 1025
  - type casting, 190–192
  - value returned, 181–250, 804–807
  - virtual, 864–876
  - void*, 252–259
- G**
- Generic algorithms, 991–1004
  - big-O notation, 994–995
  - container access running times, 995–996
  - container modifying, 1001–1002
  - nonmodifying sequence, 997–1001
  - running times, 991–996
  - set, 1003–1004
  - sorting, 1004
  - templates for, 991–1004
- get function, 338–341
- getline function, 465–466, 475–476, 478–479
- Global named constants, 221–224
- Global scope, 226–227
- Global variables, 223–224, 518
- graph function, 407
- Greater than comparison operator (>), 78
- Greater than or equal to comparison operator (>=), 78, 80
- H**
- Handling exceptions, 894
- Hard disks, 6
- Hardware
  - computer systems and, 2–7, 32
  - input/output devices, 3
  - main memory, 3–5
  - processor (CPU), 3, 6–7
  - secondary memory, 6
- Header files (< >), predefined functions, 184–186
- Hierarchy of structures, 549
- High-level languages, 8–9
- I**
- Identifiers, variables, 42–44
- if-else* statements, 75–82, 120–128
  - Boolean expressions for, 77–79
  - braces { } used with, 82, 121–123
  - branching mechanisms, 75–82, 120–128
  - comparison operators for, 77–82
  - compound statements and, 82
  - dangling *else* problem, 121–123
  - indenting, 120–121, 123–125
  - multiway branches, 123–128
  - nested, 120–123
- #ifndef* directive, 716–718
- ifstream*, 308–309, 318
- Implementation files, ADT, 592–597, 708–715, 718–719
- Implementation phase, 15
- #include* directive, 21, 25–26, 52–53, 184–186, 309, 318, 716–718
  - C++ programming and, 21, 25–26
  - directive notation (#) for, 25
  - file I/O, 309, 318, 329
  - header files and, 184–186
  - #ifndef* directive and, 716–718
  - manipulator functions and, 329
- output and, 52–53
- predefined functions and, 186–187
- preprocessors for, 186
- separate compilation and, 716–718
- Increment operators (++), 87–91, 141–144, 960–961, 967–969, 971
- Indentation, C++ programming and, 93
- Indenting branching statements, 120–121, 123–125
- Index (subscript) of arrays, 379, 383–384
- Indexed variables, 379–386, 389–391, 426, 431
  - arguments to functions, as, 389–391
  - arrays and, 379–386
  - commas (,) between, 431
  - declaration of, 379–384
  - functions and, 389–391
  - illegal range of, 383–384
  - initializing, 386
  - multidimensional arrays, 426, 431
  - square brackets [ ] used for, 378–379, 431
- Infinite loop statements, 87–91, 152
- Infinite recursion, 799
- Information hiding, 205, 597. *See also* Procedural abstraction
- Inheritance, 17, 598–603, 833–892
  - ancestor class, 844
  - assignment (=) operators used for, 860–861
  - base class, 834, 836–837, 848–850, 858
  - child class, 600, 834, 844
  - class hierarchy, 599–600
  - colon (:) used for, 600–601
  - constructors used in, 845–848

- copy constructors used in, 860–861
- derived classes and, 598–603, 834–836, 837–845, 854–856, 860–861
- descendants, 844
- destructors and, 860–861
- function signature, 857
- functions not inherited, 850, 859–860
- member functions, 845, 850–852, 853–856
- parent class, 600–601, 834, 844
- polymorphism and, 862–876
- private members and, 848–850
- protected qualifier, 850–852
- redefining functions, 853–856
- Initialization, 48–49, 145–146, 386, 454–455, 551, 576–583
  - arrays, 386, 454–455
  - C strings, 454–455
  - constructors for, 576–583
  - declaration and, 48–49
  - objects, 576–583
  - structures, 551
  - variables, 48–49, 145–146, 386, 454–455
- Inline functions, 1026
- Input, 3, 21–23, 50, 56–58, 157–160, 306–312, 343–349
  - character data, 343–349
  - cin statements for, 21–23, 56–57
  - computer hardware
    - devices, 3
  - echoing, 58
  - extraction operator (>>) for, 309
  - get function, 338–341
  - loops, design for ending, 157–160
  - member functions for, 343–348
  - new\_line( ), 343–345, 347–348
  - new-line character (\n) and, 345–346
  - put function, 341–342
  - putback function, 342–343
  - reading files, 308–309
  - streams, 50, 306–312
- Input iterators, 972
- Input/output (I/O), 50–59, 305–376, 464–466, 475–477, 1020–1021
  - arguments (parameters) and, 332, 348–349
  - C++ programming and, 50–59
  - C strings, 464–466
  - character, 338–360
  - cin (input) statements, 56–57
  - cout (output) statements, 50–52
  - decimal points for formatting numbers, 55–56
  - designing, 58
  - double statements, 55–56
  - end of files (eof), 332–335, 353–354
  - escape sequences, 53–55
  - files, 306–323, 332–337
  - flags, 325–327
  - formatting, 323–338
  - functions, 323–349, 353–354, 1020–1021
  - getline function, 475–476
  - #include directive, 52–53
  - manipulators, 329
  - namespaces, 52–53, 335–336
  - new\_line function, 343–344, 346–348
  - new-line instruction (\n), 54–55, 58, 340–341, 345–346
  - predefined character functions, 356–360
  - streams, 50, 305–376
  - string class for, 475–477
  - using directive, 52, 335–336
- Insertion operator (<<), 310, 316–318, 329, 464, 650–658
- in\_stream, 307, 308–309, 312–314, 318
- int, 21, 44, 60–61, 63, 70–72, 116–118, 378–382, 489–491
  - arithmetic operators and, 70–72
  - array declaration using, 378–382
  - Boolean expressions and, 116–118
  - enumerated types, 119–120
  - numeric data type, 44, 60–61, 63
  - unsigned type, 490–491
  - value conversion, 116–118
  - variable declaration using, 21, 44, 378–382, 489–491
  - vector declaration using, 489–491
- Integers, 21, 60–62, 190–192
  - data values, 60–62
  - type casting by division, 190–192
  - variables, 21
- Interface files, ADT, 592–593, 705–707, 713, 723, 724
- ios::fixed flag, 325–327
- ios::left flag, 327
- ios::right flag, 327
- ios::showpoint flag, 325–327
- ios::showpos flag, 327
- iostream library, 25
- isalpha function, 359
- isdigit function, 359
- islower function, 359
- isspace function, 358–359
- isupper function, 359

- Iterators, 84–87, 157, 159, 755, 802–803, 817–818, 959–973
  - auto, variable declaration using, 964
  - bidirectional, 966–969
  - compiler problems, 964–965
  - constant, 970–971
  - decrement operators (--)
    - for, 967–969
  - dereferencing (\*) operator
    - for, 964–965
  - forward, 969
  - increment operators (++)
    - for, 960–961, 967–969, 971
  - input, 972
  - loop mechanisms and, 84–87, 157, 159
  - mutable, 970
  - operators for, 960–961
  - output, 973
  - pointers as, 755
  - random access, 966–969
  - recursion compared to, 802–803
  - recursive program version, 817–818
  - reverse, 971–972
  - templates for, 959–973
  - types of, 966–971
  - using directives for, 959–960
  - vectors and, 961–965
- L**
- Languages, 8–11, 18–19
  - assembly, 8
  - C++ programming, 18–19
  - compilers for translation of, 9–11
  - computer programs and, 8–9
  - high-level, 8–9
  - linker programs for, 9–11
  - low-level, 8
  - machine, 8–9
  - program translation of, 8–11
- Last-in/first-out (LIFO) data structure, 766, 801–802
- Late (dynamic) binding, 863–869
- Leaf nodes, 762
- length function, 480–481
- Less than comparison operator (<), 78
- Less than or equal to comparison operator (<=), 78
- Lexicographic order, 482
- Line breaks, C++ programming, 24, 26, 202
- Linear running time, 995
- Linked lists, 739–787, 974. *See also* Containers
  - arguments, as, 747
  - assignment (=) operators
    - used with, 757–758
  - classes and, 762–765
  - data structures, as, 739–742
  - doubly, 760–761, 974
  - dynamic data structures in, 740, 757–758
  - head of, 746–750
  - inserting nodes in, 747–749, 755–757
  - losing nodes, 750–751
  - middle, 755–757
  - Node class, 762–765
  - nodes and, 740–742, 747–750, 755–757
  - pointers and, 739–787
  - queues and, 771–776
  - removing nodes from, 755–757
  - searching, 751–754
  - singly, 974
  - stacks, 765–766
- Linker programs, 9–11
- List headed-by-size loop termination, 157
- Local variables, 135–137, 218–229, 270–271
  - block scope, 135–137, 226–227
  - call-by-value parameters as, 224–226, 270–271
- functions and, 218–229, 270–271
- global constants and, 221–223
- global scope, 226–227
- global variables and, 223–224
- inadvertent, 270–271
- namespaces and, 227–229
- scope of, 220–221, 226–227
- Logic errors, 30–31
- long data type, 62–63
- Loop mechanisms, 84–91, 98, 112–120, 139–155, 208–211
  - ask-before-iterating technique for, 157, 159
  - body, 84–86
  - Boolean expressions for, 84–87, 112–120
  - braces { } for execution of, 84–86
  - break* statement for, 153–154
  - count-controlled, 158
  - debugging, 162–164
  - decrement operators (--), 87–91, 143–144
  - design choices, 150
  - do-while* statements, 87–91, 139–140, 154
  - ending input loops, 157–160
  - exit-on-flag termination, 159
  - flags, 159
  - flow of control using, 84–91, 98, 139–155
  - for* statements, 144–150, 154
  - increment operators (++), 87–91, 141–144
  - infinite, 87–91, 152
  - iteration, 84–87, 157, 159
  - list headed-by-size termination, 157
  - nested, 154, 160–161, 208–211
  - procedural abstraction and, 208–211

- products obtained using, 156–157
- semicolons (;) and, 149–150
- sentinel value, 158
- sums obtained using, 155–156
- uninitialized variables and, 152
- while* statements, 84–91, 139–144, 153–154
- zero times body execution, 87, 141
- Low-level language, 8
- M**
- Machine language, 8–9
- main()* function, 25
- Main memory, 3–5
- Mainframe computer systems, 2
- Manipulator functions, 329
- map* class, 983–990
- Member functions, 312–314, 338–354, 465–466, 480–483, 554–558, 570–574, 576–588, 818–821, 845, 848–856
- at, 480–481
- accessor functions and, 567–568
- BankAccount* class examples of, 570–574
- blank spaces and, 338–339
- C strings, 465–466
- character I/O and, 338–354
- classes and, 312–314, 554–558, 570–574, 576–588
- constructors, 576–588
- definition of, 554–558
- dot* (.) operator used for, 313, 557–558
- eof*, 353–354
- exit*, 315
- fail*, 314
- get*, 338–341
- getline*, 465–466
- inheritance and, 845, 850–852
- length, 480–481
- mutator functions and, 567–568
- new\_line()*, 343–345, 347–348
- new-line character (\n) and, 338–360, 345–346
- objects and, 312–314
- private*, 559–568, 848–850
- protected*, 850–852
- public*, 559–568
- put*, 341–342
- putback* function, 342–343
- recursion and, 818–821
- redefinition of, 853–856
- scope resolution (::) operator used for, 557–558
- stream I/O using, 312–314
- string* class use of, 480–483
- Member names, structures, 543, 545–546
- Member values, structures, 543, 546
- Member variables, structures, 543, 545–547, 550
- Memory, 3–6, 40–42, 262–264, 382–383, 393–394, 516–518
- addresses, 4–5
- array declaration and, 382–383
- array parameters, 393–394
- bits (binary digits), 4
- bytes, 4–5
- call-by-reference parameters and, 262–264
- computer hardware components, 3–6
- delete* operator for, 517–518
- dynamic variables, 516–518
- files, 6
- freestore, 516–517
- locations, 4–5, 41–42, 262–264
- main, 3–5
- management, 516–518
- pointers for, 516–518
- random access (RAM), 6
- secondary (auxiliary), 3, 6–7
- sequential access of, 6
- storage as, 6
- variables and, 40–42
- Menus, 133–134
- program choices using, 133–134
- switch* statements for, 133–134
- Messages, errors, 30–31
- Monitor, computer output device, 3
- Multidimensional arrays, 425–431, 530–532
- commas (,) between indexes, 431
- declarations for, 426–427
- delete* [ ] operator and, 530–531
- dynamic, 530–532
- indexed variables and, 426, 431
- parameters, 426–427
- size of, 426–427
- square brackets [ ] used for, 427, 431
- two-dimensional example of, 427, 531–532
- Multiplication operator (\*), 70
- Mutable iterators, 970
- Mutator functions, 567–568
- N**
- Names, 42–45, 49, 95–97, 207–208, 221–224, 232–238, 308–310, 318, 518–520
- constants, 95–97, 221–224
- data types, 44–45
- external file, 310
- files, 308–310, 318
- formal parameters, 207–208
- functions and, 221–224, 232–238



- Names (*continued*)
    - global constants, 221–224
    - identifiers, 42–44
    - overloading functions, 232–238
    - pointer types, 518–520
    - procedural abstractions, 207–208
    - streams, 308–310, 318
    - typedef* function, 518–520
    - variables, 42–45, 49, 308
  - Namespaces, 52–53, 186, 227–229, 335–336, 719–732
    - classes and, 719–732
    - creating, 721–723
    - file I/O and, 335–336
    - global, 732
    - local variables and, 227–229
    - names for, 724–726, 731
    - output and, 52–53
    - qualifying names, 724–726
    - stream I/O and, 335–336
    - unnamed, 726–732
    - using directives for, 52–53, 186, 228–229, 335–336, 719–721, 724–726
  - Nesting, 120–123, 137, 154, 160–161, 208–211, 916
    - blocks, 137, 916
    - braces { } used for, 121–123, 137–138
    - break* statement in, 154
    - dangling *else* problem, 121–122
    - function calls and, 208
    - if-else* statements, 120–123
    - indenting statements, 120–121
    - loops, 154, 160–161, 208–211
    - multiway branches, 120–123, 137
    - procedural abstraction and, 208–211
    - scope of the block for, 137
    - statements, 120–123
    - try-catch* blocks, 916
  - Network computer systems, 2
  - new operator, 513–515, 524–527
    - dynamic arrays and, 524–527
    - pointers using, 513–515
  - new\_line()* function, 343–345, 347–348
  - New-line instruction (*\n*), 23, 53, 54–55, 58, 338–360, 345–346
    - C++ programming and, 23
    - endl* used in place of, 54–55
    - input and, 345–346
    - member functions and, 338–360
    - output and, 54–55
  - Node class, 762–765
  - Nodes, 740–742, 747–750, 755–765
    - arrow (*->*) operator used with, 742, 744
    - binary trees and, 761–762
    - doubly linked lists, 760–761
    - head (front) of lists, inserting at, 747–749
    - inserting to lists, 747–749, 755–757
    - leaf, 762
    - linked lists and, 740–742, 747–750, 755–757
    - lost, 750–751
    - middle of lists, inserting and removing, 755–757
    - NULL constant used in, 742–744
    - pointer variables and, 741–742
    - removing from lists, 755–757
    - root, 762
    - searching linked lists using, 751–754
    - structures, 740–742
    - trees and, 761–762
  - Nonmember functions, 624–628
  - Nonmodifying sequence algorithms, 997–1001
  - Not equal to comparison operator (*!=*), 78–79
  - Not operator (*!*), 79, 113, 118
  - Null (*/0*) character, 453–454, 456
  - NULL constant, 742–744
  - Null statements, 150
  - nullptr*, in C++ 11 programming, 745
  - Number formatting, decimal points for, 55–56
  - Number-to-C string conversions, 466–470
  - Numeric calculations, 143–146, 155–157. *See also* Arithmetic operators
    - for* loop statements for, 143–146
    - loop design for, 155–157
    - products, 156–157
    - sums, 155–156
  - Numeric data values, 44, 60–64
- ## O
- Object code, 9–11, 26–27
  - Object-oriented programming (OOP), 16–17
    - classes, 17
    - encapsulation, 17
    - inheritance, 17
    - polymorphism, 17
    - program design using, 16–17
  - Objects, 312–315, 554, 566, 569, 576–588
    - assignment operator (*=*) used with, 569
    - classes and, 312–315, 554, 566
    - constructors for, 576–588
    - file I/O and, 312–315
    - initialization of, 576–583
    - member functions, 312, 576–588
    - public* and *private* specification, 566
    - streams and, 312–315

- Off-by-one error, 162
  - ofstream*, 308–309, 318
  - Op* operator, 74
  - open* function, 309–310, 318
  - Operating systems, computer software for, 7
  - Operators, 69–74, 77–82, 87–91, 112–120, 309, 316–318, 464, 643–658, 1016–1017, 1032–1033
    - arithmetic, 69–74, 112–116
    - Boolean expressions, 77–79, 112–120
    - comparison, 77–82
    - decrement (--), 87–91
    - extraction (>), 309, 316–318, 464, 650–658
    - increment (++), 87–91
    - insertion (<<), 310, 316–318, 329, 464, 650–658
    - overloading, 643–658, 1032–1033
    - precedence, 114–115, 1016–1017
    - unary, 87, 649–650
  - Or operator (||), 78, 80, 112–116
  - Output, 3, 21–23, 50–56, 58, 306–312, 323–338
    - computer hardware devices, 3
    - cout* statements, 21–23, 50–52
    - decimal points for formatting numbers, 55–56
    - double* statements, 55–56
    - escape sequences, 53–55
    - flags, 325–327
    - formatting functions, 323–338
    - insertion operator (<<) for, 310
    - manipulators, 329
    - new-line instruction (\n) for, 54–55, 58
    - streams, 50, 306–312, 323–338
    - writing files, 308–310
  - Output iterators, 973
  - out\_stream*, 307, 308–310, 312–314, 318, 325–326
  - Overloading, 232–238, 578, 643–658, 821, 1027–1028, 1032–1033
    - array index, 1027–1028
    - constructors, 578
    - extraction operator (>), 650–658
    - function names, 232–238
    - insertion operator (<<), 650–658
    - operators, 643–658, 1032–1033
    - recursion compared to, 821
    - type conversion and, 238, 647–649
    - unary operators, 649–650
  - Overriding functions, 869
- ## P
- Parameters, 197–201, 207–208, 224–226, 259–266, 346–349, 391–397, 414, 426–427, 460, 519–520, 638–642
    - arguments and, 197–198, 200–201, 265–268, 348–349, 460
    - array, 391–396
    - arrays and, 391–397, 414, 426–427, 460
  - C string, 460
  - call-by-reference, 259–266, 519–520
  - call-by-value, 197–198, 224–226
  - character I/O, 346–349
  - const* modifier, 394–397, 638–642
  - constant, 638–639
  - constant array, 395
  - formal, 197–201, 207–208, 224–226, 414
  - friend* functions and, 638–642
  - function arguments and, 414
  - function calls using, 197–198, 259–260
  - function declarations using, 199–201
  - function subtasks using, 259–266
  - local variables and, 224–226, 270–271
  - memory locations, 262–264, 393–394
  - mixed lists, 268–271
  - multidimensional arrays, 426–427
  - names, 207–208, 262
  - pointers, 519–520
  - procedural abstraction and, 207–208
  - programmer-defined functions, 197–201
    - size of arrays and, 394, 414
    - stream versatility, 346–347
  - Parent class, 600–601, 834, 844
  - Parentheses ( ), 71, 78–79, 84–86, 130
  - Partially filled arrays, 411–413
  - Personal computer (PC), 2
  - Pointer variables, 521–523, 527
  - Pointers, 507–540, 674–675, 739–787, 1029–1031
    - addresses, 509–511
    - ampersand (&) symbol and, 511
    - arithmetic performed on, 528–529
    - arrow (->) operator used with, 742, 744
    - assignment operator (=) and, 511–512, 514, 757–758
    - asterisk (\*) used for, 509–512
    - automatic variables, 518
    - call-by-reference parameters for, 519–520
    - call-by-value parameters, 674–675
    - dangling, 522–523

- Pointers (*continued*)
  - declaration of, 509–510
  - delete* operator, 517–518, 524–527, 530–531
  - dereferencing (\*) operator
    - for, 510–511
  - destructors and, 674–675
  - dynamic arrays and, 521–527, 530–532, 674–675
  - dynamic variables and, 513, 516–518, 740, 757–758
  - freestore, 516–517
  - iterators, used as, 755
  - linked lists and, 739–787
  - memory management for, 516–518
  - names, 518–520
  - new* operator, 513–515
  - nodes, 740–742, 747–750, 755–757
  - NULL constant assigned to, 742–744
  - queues and, 771–776
  - stacks and, 765–766
  - static variables, 518
  - structures containing, 741–742
  - this*, 1029–1031
  - trees and, 761–762
  - typedef* function, 518–520, 530
  - variables and, 508–520, 521–523, 527, 741–742
- Polymorphism, 17, 862–876
  - destructors made virtual for, 875–876
  - errors, 874–875
  - late (dynamic) binding, 863–869
  - overriding functions, 869
  - virtual functions and, 864–876
- pop* function, 770
- Postconditions, 275–281
- pow* function, 187–188
- Precedence rules, 114–115
- Preconditions, 275–281
- Predefined functions, 183–192, 356, 358–360, 457–460, 467–468
  - abs*, 186–187
  - absolute values, 186–187
  - arguments, 183–184, 459–460
  - C string, 457–460, 467–468
  - calls (invocations), 183–188
  - character I/O data, 356, 358–360
  - fabs*, 187
  - header files (< >) and, 184–186
  - #include* directives, 184–186
  - isspace*, 358–359
  - parentheses ( ) and, 184, 191
  - pow*, 187–188
  - random number generation
    - using, 188–189
  - sqrt*, 183–185, 187
  - srand*, 187, 189
  - strcmp*, 457–460
  - string-to-number conversions, 467–468
  - strncpy*, 457–460
  - toupper* and *tolower*, 358–360
  - type casting using, 190–193
  - using directive, 186
  - value returned, 183, 358–360
- priority\_queue* class, 979–983
- private members, 559–568, 593, 597, 624, 848–850
  - abstract data types (ADT) and, 593
  - accessor functions and, 567–568
  - classes using, 559–568
  - friend function access to, 624
  - inheritance and, 848–850
  - mutator functions and, 567–568
- public members and, 559–568
- Problem-solving phase, 15, 211–212, 277–278, 398–399
- Procedural abstraction, 204–217, 273–281
  - algorithm design for, 212–213, 278
  - black box analogy, 204–207
  - case study: Buying Pizza, 211–217
  - case study: Supermarket Pricing, 276–281
  - coding, 213–214, 278–280
  - functions calling functions, 273–275
  - functions returning values, 204–217
  - information hiding, 205–206
  - nested loops and, 208–211
  - parameter names and, 207–208
  - postconditions, 275–281
  - preconditions, 275–281
  - problem analysis, 211–212, 277–278
  - program testing, 214–217, 281
  - pseudocode for, 213, 217
  - subfunctions using, 273–281
- Processor (CPU), computer component, 3
- Programmer role, 19
- Programmer-defined functions, 193–203
  - arguments and, 197–198, 200–201
  - body, 196
  - bool* values, returning, 199
  - branching statements and, 203
  - call-by-value parameters, 197–198
  - calls, 196–198, 203
  - declaration, 193, 195–196, 199–201

- definitions, 193–194, 196–197, 201–203
- headers, 196, 200
- parameters, 197–201
- return* statements, 196–197, 202
- spacing and line breaks in, 202
- syntax of, 201–202
- value returned, 196, 199
- Programming, 12–17, 588–597. *See also* C++ programming; C++11 programming
  - abstract data types (ADT), 588–597
  - algorithms, 12–16
  - implementation phase, 15
  - object-oriented (OOP), 16–17
  - problem-solving phase, 15
  - program design for, 15–17
  - software life cycle, 17
- Programs, 2, 7–11, 13–16, 27–31
  - algorithms for, 13–16
  - compiler, 9–11, 27–30
  - computer software as, 2, 7–8
  - debugging, 29–31
  - design of, for programming, 15–17
  - executing, 8
  - high-level languages for, 8–9
  - implementation design phase, 15
  - language translation and, 8–11
  - linker, 9–11
  - logic errors, 30–31
  - object code
  - problem-solving design phase, 15
  - run-time errors, 30
  - running, 8–11
  - source code, 9
  - syntax error, 30
  - testing, 27–31
- protected* members, 850–852
- Pseudocode, 213, 217
- public members, 559–568
  - accessor functions and, 567–568
  - classes using, 559–568
  - mutator functions and, 567–568
- put function, 341–342
- putback function, 342–343
- Q**
- Quadratic running time, 995
- queue class, 979–983
- Queues, 771–776
- R**
- Random access iterators, 966–969
- Random access memory (RAM), 6
- Random number generation, 188–189, 1024
- Ranged for, using with containers, 990
- Reading files, 308–309
- Recursion, 789–831
  - base (stopping) cases, 798
  - case study: Binary Search, 810–818
  - case study: Vertical Numbers, 791–797
  - checking program for, 816–817
  - design techniques, 809–810
  - efficiency of, 817
  - ending, 797–798
  - function definition, 791
  - functions, 791–807
  - infinite, 799
  - iteration compared to, 802–803
  - iterative version of, 817–818
  - last-in/first-out (LIFO) data structure, 801–802
  - member functions as, 818–821
  - overloading compared to, 821
  - returning values, 804–807
- stacks for, 800–802
- tasks, functions for, 791–803
- tracing recursive calls, 794–797
- values, functions for, 804–807
- void* functions and, 810
- Remainder operator (%), 70–71
- reserve function, 494
- resize function, 494
- Rethrowing exceptions, 918
- return* statements, 21, 196–197, 202, 255–259
  - C++ programming and, 21
  - functions and, 196–197
  - parentheses ( ) for, 197
  - programmer-defined functions, 196–197
  - void* functions using, 255–259
- Returning values, *see* Value returned
- Reverse iterators, 971–972
- Robust input, C strings, 468, 470–471
- Root node, 762
- Running programs, 8–11, 26–28
- Running times, 991–996
- Run-time errors, 30
- S**
- scale function, 402–407
- Scientific notation, 61–62
- Scope, 137, 220–221, 226–227
  - block, 137, 226–227
  - global, 226–227
  - local, 220–221, 226–227
  - variables, 220–221, 226–227
- Scope resolution operator (::), 557–558
- Searching arrays, 414–416
- Searching linked lists, 751–754
- Secondary (auxiliary) memory, 3, 6–7
- Selection sort, 417–418
- Semicolons (;), 24, 146, 149–150, 547

- Sentinel value, loop design and, 158
- Sequential access, memory and, 6
- Sequential containers, 974–979
- set algorithms, 1003
- set class, 983–990
- setf function, 325–327
- setprecision manipulator, 329
- setw manipulator, 329
- short data type, 63
- Short-circuit evaluation, 115
- Single quotes ( ' ') for constant characters, 65
- Single-precision numbers, 61
- Size (number of elements), 379–382, 394, 411–414, 426–427, 493–494, 668–671
  - array parameters, 394
  - arrays, 379–382, 394, 411–414, 426–427
  - capacity compared to, 492
  - const modifier for, 376–382
  - constructors and, 668–671
  - declared, 379
  - dynamic arrays, 668–671
  - function arguments and, 414
  - multidimensional arrays, 426–427
  - partially filled arrays, 411–413
  - resize function, 494
  - vectors, 493–494
- Software, 2, 7–8, 17, 715
  - abstract data types (ADT), 715
  - computer operating systems, 7–8
  - life cycle, 17
  - programs, 2
  - reusable components, 715
- Sorting algorithms, 1004
- Sorting arrays, 417–423
- Source code, 9
- Spacing, 24, 26, 202, 338–339
  - C++ programming, 24
  - character I/O and, 338–339
  - function definition and, 202
- sqrt function, 183–185, 187
- Square brackets [ ], 378–379, 392–393, 427, 431, 492
- srand function, 187, 189
- Stack class, 766–770, 979–983
- Stacks, 765–766, 800–802, 801–802
  - empty, 770
  - implementation of, 768–770
  - last-in/first-out (LIFO) data structure, 766
  - linked lists as, 765–766
  - overflow, 802
  - pointers and, 765–766
  - pop function, 770
  - recursion and, 800–802
- Standard Template Library (STL), 963–1013. *See also* Templates
- Statements, 21–26, 33
  - C++ programming instructions, 21–26, 33
  - cin (input), 21–23
  - cout (output), 21–23
  - direction arrows (<< >>), 21
  - directives #, 25
  - executable, 21–26
  - #include directive, 21, 25, 26
  - new line (\n), 23
  - return, 25–26
  - semicolon (;), 24
- Static variables, 518
- static\_cast<double>, 190–192
- std namespace, 335–336
- Stepwise refinement, 182–183
- Storage, memory as, 6
- strcat function, 459–460
- strcmp function, 457–460
- strcpy function, 457–461
- Streams, 50, 305–376
  - appending to a file, 320–322
  - arguments to functions, as, 332
  - character I/O and, 338–360
  - cin as, 307
  - classes and, 312–315
  - cout as, 307
  - declaring, 308–310
  - default arguments, 348–349
  - fail function, 314
  - file names and, 308–310, 318
  - files and, 306–338
  - flags and, 325–327
  - formatting functions, 323–328
  - ifstream, 308–309, 318
  - input/output (I/O), 50
  - in\_stream, 307, 308–309, 312–314, 318
  - manipulator functions for, 329
  - member functions and, 312–314, 338–349
  - namespaces and, 335–336
  - objects and, 312–315
  - ofstream, 308–309, 318
  - output, formatting using, 323–338
  - out\_stream, 307, 308–310, 312–314, 318, 325–326
  - parameters, 346–349
  - using directives and, 335–336
  - variables as, 308
- string class, 66–68, 472–488
  - <string> library, 472, 474
  - characters, 66–68
  - comparison operators and, 482, 487
  - concatenation (+), 66–67, 472
  - constants converted to, 473
  - data types and, 66–68
  - default constructor for, 473–474
  - double quotes ( " ") for characters, 64–65
  - getline function, 475–476, 478–479
  - input/output (I/O) using, 475–477

- lexicographic ordering of, 482
- member functions, 480–483
- object-to-C string conversion, 487–488
- palindrome testing program
  - example, 484–487
- variable declaration, 66–68
- whitespace characters and, 68
- String functions, 1023
- String values, 456–458, 668–671
  - C strings, 456–458
  - dynamic arrays, 668–671
  - implementation, 671–673
  - size of, 668
- String variables, 322–323
- stringvar class, 668–671
- strlen function, 459
- strncat function, 459
- strncmp function, 459
- strncpy function, 457, 459
- Structures, 542–550, 575, 740–742
  - braces { } for, 543, 547
  - classes compared to, 542–550, 575
  - diverse data of, 542–547
  - dot operator (.) for, 545, 550
  - functional arguments, as, 548–549
  - hierarchy of, 549
  - initializing, 551
  - linked lists and, 740–742
  - member names, 543, 545–546
  - member values, 543, 546
  - member variables, 543, 545–547, 550
  - nodes as, 740–742
  - pointer variables for, 741–742
  - semicolons (;) for, 547
  - value, 543
- Stubs, function testing using, 284–286
- Subexpressions, 115
- Subtasks, 251–303
  - assert macro, 290–291
  - call-by-reference parameters, 259–266
  - debugging functions, 282–287
  - functions for, 251–303
  - procedural abstraction, 273–281
  - testing functions, 282–287
  - void functions, 252–259
- Subtraction operator (-), 70
- switch statements, 128–135
  - break statements, 131–133
  - menus, 133–134
  - multiway branching, 128–135
- Syntax, 30, 45, 939–941
  - class templates for, 939–941
  - errors, 30
  - variable declaration and, 45
- T**
- Tasks, recursive functions for, 791–803
- Templates, 925–956, 963–1013
  - algorithm abstraction, 926–938
  - class syntax, 939–941
  - containers, 973–990
  - data abstractions, 939–948
  - function definition, 937–938
  - generic algorithms, 991–1004
  - iterators, 959–973
  - Standard Template Library (STL), 963–1013
  - type definitions, 942
- Terminal, computer output device, 3
- Testing programs, 27–31, 214–217, 281–287, 407
  - boundary values, 281
  - compiling and running programs, 27–29
  - debugging and, 29–31
  - drivers, 282–284
  - error messages, 30–31
  - functions, 214–217, 281
  - input, 281
  - logic errors, 30–31
  - procedural abstraction and, 214–217, 281
  - program testing, 214–217, 281
  - run-time errors, 30
  - scale function, 407
  - stubs, 284–286
  - syntax errors, 30
  - warning messages, 30
- Text files, editing, 355–357
- this pointer, 1029–1031
- Throw list, 911–913
- throw statement, 898–900, 909–911
- Throwing exceptions, 894, 909–911, 914–916
- Top-down design, 182–183, 398–409
- toupper and tolower functions, 358–360
- Tracing recursive calls, 794–797
- Tracing variables, 162–163, 288
- Trees, data structures as, 761–762
- Trigonometric functions, 1025
- Trivial exceptions, 909
- true/false values, 66, 116. *See also* Boolean expressions
- Truth tables, 112–114
- try-catch blocks, 916
- try-throw-catch mechanism, 898, 901–903
- Two-dimensional arrays, 427, 531–532
- Type casting, 190–192
- Type name, variables, 44–45
- typedef function, 518–520, 530
- U**
- Unary operators, 87, 649–650
- Uninitialized variables, 47–49, 152
- unsigned int type, 490–491
- User role, 19
- using directive, 52, 186, 228–229, 335–336, 719–721, 724–726, 959–960



## V

Value returned, 183, 196–197,  
199, 202, 358–360,  
804–807, 809–810  
*bool* statements, 199  
character data, 358–360  
predefined functions, 183  
programmer-defined func-  
tions, 196–197  
recursion, 804–807  
*return* statements,  
196–197, 202  
*toupper* and *tolower*, func-  
tions for, 358–360  
Values, recursive functions for,  
804–807  
Variables, 21–23, 40–50,  
60–74, 135–137, 143–144,  
147–149, 152, 162–163,  
218–229, 288, 308,  
378–386, 389–391, 426,  
431, 453–460, 489–492,  
508–520, 521–523, 527,  
664–667, 668–671  
arithmetic operators for, 69–72  
arrays and, 378–386,  
389–391, 453–460,  
521–523, 527, 664–667,  
668–671  
assignment statements,  
45–49, 69, 511–512  
asterisk (\*) used for, 509–512  
automatic, 518  
blocks and, 135–137  
C strings, 453–460  
*cin* (input) statements, 21–23  
class members, 664–667  
*cout* (output) statements,  
21–23  
data types, 44–45, 60–74  
declaration of, 21–23,  
44–45, 48–49, 64,  
66–68, 147, 378–382,  
453–454  
dereferencing (\*) operator  
for, 510–511  
dynamic, 513, 516–518

dynamic arrays and,  
521–523, 527, 668–671  
equal sign (=) for, 22  
*for* statements for, 147–149  
function and, 218–229  
global, 223–224, 226–227,  
518  
identifiers, 42–44  
increment/decrement opera-  
tors for, 143–144  
indexed, 379–386, 389–391,  
426, 431, 489  
initializing, 48–49,  
145–146, 386, 454–455  
integers as, 21, 44–45  
local, 135–137, 218–229  
loop mechanisms and,  
135–137, 143–144,  
147–149, 152, 162–163  
memory locations, 41–42,  
382–383  
naming, 42–45, 49  
*new* operator for, 513–515  
null (/0) character and,  
453–454, 456  
pointers, 508–520, 521–523,  
527  
scope, 220–221, 226–227  
square brackets [ ] used for,  
378–380, 492  
static, 518  
streams as, 308  
*string*, 66–68, 668–671  
syntax for, 45  
tracing, 162–163, 288  
type name, 44–45  
uninitialized, 47–49, 152  
values, 22, 45–48, 162–163,  
489–492  
vectors, 489–492  
Vectors, 489–494, 961–965  
assignment operator (=) for,  
493  
*capacity*( ) function,  
493–494  
capacity of, 493–494  
constructor, 492

declaring variables, 489–490  
efficiency of, 493–494  
indexed variables, 489  
iterators for, 961–965  
*reserve* function, 494  
size of, 493  
square brackets [ ] used for,  
492  
*unsigned int* type, 490–491  
variable values, 489–492  
Virtual functions, polymor-  
phism and, 864–876  
*void* functions, 252–259, 810  
C++ definition, 252–254  
calls, 253–254  
recursion and, 810  
*return* statements in,  
255–259  
syntax, 253

## W

Warning messages, 30  
*while* loop statements, 84–91,  
139–144, 153–154  
braces { } for execution of,  
84–86  
*break* statement for, 153–154  
increment and decre-  
ment operators, 87–91,  
141–144  
infinite, 87–91  
nested, 154  
syntax of, 86, 140  
zero times body execution,  
87, 141  
Whitespace characters, 68, 358  
*width* function, 328  
Workstation, 2  
Writing abstract data types  
(ADT), 591–592  
Writing files, 308–310

## Z

Zero times loop body  
execution, 87, 141  
Zeros leading in number  
constants, 636







