Symbols and Numerics

-- (decrement) unary operator
defined, 50–51
looping statements, 74–75
order of precedence, 48
prefix and postfix versions, 51, 74–75
- (negative) unary operator
defined, 50
order of precedence, 48
- (subtraction) binary operator, 48, 136
! (NOT) logical operator, 54
!= (inequality) logical operator, 54
# symbol, 154, 158
%(modulus) binary operator, 304
defined, 48
order of precedence, 48
& (AND) bitwise operator
defined, 62
test program using, 65–67
using, 64
values, 63
& (unary) pointer operator, 123–124
&& (AND) logical operator
defined, 54–55
general discussion, 53
short-circuit evaluation, 58–59
( ) parenthesis, order of precedence
and, 50
* (multiplication) binary operator,
48, 137
* (unary) pointer operator, 123, 125–126,
193–194
** wildcards, 150
. (dot) operator, 304
/ (division) binary operator, 48
// (double slash) comment
characters, 28
/* */ comment characters, 28
: : (scope resolution) operator, 183–184,
304, 375
; (se colons), 28
?: (ternary) operator, 304
\ \ (backslash) character, 42–43
^ (XOR) bitwise operator
defined, 62
test program using, 65–67
values, 64
| (OR) bitwise operator
defined, 62
test program using, 65–67
values, 63
|| (OR) logical operator
defined, 54–55
short-circuit evaluation, 58
~ (NOT) bitwise operator
defined, 62
destructors, 221
using, 64
values, 63
+ (addition) binary operator, 48, 136
+ (positive) unary operator
defined, 50
order of precedence, 48
++ (increment) unary operator
defined, 50–51
iterators, 377
looping statements, 74
order of precedence, 48
prefix and postfix versions, 51
reasons for creating, 51
< (less than) logical operator, 54–55, 371
<< (insertion) operator, 304, 315–316, 371
<= (less than or equal to) logical
operator, 54–55
= (assignment operator), 30, 51–52, 54,
371, 375
== (equality) logical operator
assignment operator versus, 54
defined, 54
list class, 375
string class, 371
-> (arrow) operator, 194, 304
> (greater than) logical operator, 54–55, 371
>= (greater than or equal to) logical
operator, 54–55
>> (extraction) operator, 304, 315–316, 371
\0 (null character), 42–43, 114–115
\0nn (octal character), 42–43

• A •

abstract classes
concept of, 295–297
passing, 298
subclasses of, 298
accumulator value, 81, 84, 109
addition (+) binary operator, 48, 136
Address Space Layout Randomization
(ALSR), 404
AND (&) bitwise operator
defined, 62
test program using, 65–67
using, 64
values, 63
AND (&&) logical operator
defined, 54–55
general discussion, 53
short-circuit evaluation, 58–59
app flag, 318
append( ) function, 371
arguments
default values for, 101–102
defined, 93
functions with, 95–97
functions with multiple, 97
optional nature of, 94
passing by reference, 102–103
passing by value, 102–103
ArrayDemo sample program
displayArray( ) function, 108–109
inputValues[ ] array, 109
overview, 107–108
readArray( ) function, 108–109
sumArray( ) function, 108–109
ArrayOfStudents sample program, 191–192
arrays
accessing individual elements of, 106
applying operators to addresses,
138–139
arrays of, 112
of characters
concatenating strings, 115–117
creating, 112–113
creating strings of characters, 114–115
utilizing, 146–147
constructing, 244
correlated with pointers, 142–144
declaring, 106
defined, 105
deleting, 224
disadvantages of, 369
element of, 107–109
initializing, 110
of objects
allocating off heap, 200
constructing, 218–219
declaring, 191–192
of pointers, 145–146
range-based loops, 111–112
relationship to pointers, 136–138
size of, 106, 109
string-handling functions, 117–118
wide string-handling functions, 118–120
writing beyond range of, 110–111
arrow (->) operator, 194, 304
ASLR (Address Space Layout
Randomization), 404
assert( ) function, 165
assignment operator (=), 30, 51–52, 54,
371, 375
assignment operators
copy constructors versus, 305–306
deleting, 310
move constructor and, 312–314
order of precedence, 48
overloading, 305, 306–311
overview, 51–52
return type and value, 310
at( ) function, 371
ate flag, 318, 319
auto keyword, 45, 46
automatic variables, 110

• B •
backslash character (\), 42–43
bad( ) function, 320, 323
base 2 (binary) number system, 60
base 8 (octal) number system, 60–61
base 10 (decimal) number system, 59–60
base 16 (hexadecimal) number system, 61–62
beg flag, 330
begin( ) function, 375, 377–378
binary (base 2) number system, 60
binary flag, 318, 319
binary operators
defined, 47
order of precedence, 48
bits, defined, 60
BitTest sample program, 57, 65–66, 328
bitwise logical operators
defined, 62
general discussion, 53
test of, 65–67
using, 64
values, 63–64
bool (Boolean) variables
defined, 39, 44
list class, 375
logical operations, 55–56
string class, 371–372
boolalpha flag, 327
Boolean constants, 40
BoolTest sample program, 55–56
botnets, 382–383
branch statements, 69–71
BranchDemo sample program, 70–71
break command, 80–82, 85
BreakDemo sample program, 80–81
Budget sample programs, 3, 299
buffer overflow
avoiding, 322, 397–402
call stack, 389–393
example of, 387–389
general discussion, 386
hacking, 394–397
operating system helps, 405–406
string class, 402–404
BufferOverflow sample program, 387–389, 394, 397
bug avoidance
avoiding multiple inheritance, 416
avoiding overloading operators, 413
clear and consistent coding style, 410
commenting, 412
declaring destructors virtual, 414–416
enabling all warning and error messages, 409–410
limiting visibility of class internals, 411–412
managing heap memory systematically, 413
single-stepping programs, 413
using exceptions to handle errors, 414
building, defined, 10
bytes, defined, 60

• C •
C++
basic concepts, 9–11
comments, 27–28
declarations, 29
defined, 2
expressions
defined, 30
storing results of, 30–31
framework, 27
I/O statements, 30
as low-level standard programming language, 2
as object-oriented language, 2
as portable language, 2
programs
building, 24–25
creating projects, 19–21
downloading and installing project files, 23–24
entering code, 21–23
executing, 25–26
statements, 28–29
as strongly typed language, 34
C++ 2011 standard
0x standard, 12
automatic declarations, 45
const expressions, 160
constructing members with initializers, 237–238
default constructor, 232–233
delegating constructors, 230–231
deleting assignment operator, 310
deleting copy constructor, 258, 310
for each loops, 79
explicit keyword, 245
external template instantiations, 366
final keyword, 287
gcc compiler, 10
inheritance of constructors by subclasses, 279
initializer lists, 367
invoking constructors other than default, 244
iterators, 378, 380
move constructor, 247, 258, 312–314
nullptr constant, 145, 202, 309
override descriptor, 286–287
range-based loops, 111–112, 144
size of arrays, 106
size_type type, 372
static member functions, 267
static_assert( ) function, 165
wchar_t variables, 43
C++ 2014 standard
binary numbers, 62
const expressions, 161
constant expressions, 85
defining constants of separate, user-defined types, 161–162
gcc compiler, 10
size of arrays, 106
cache flushing, 329
CALL instruction, 390–392, 396–397
CallMemberFunction sample program, 179–182
CallStaticMember sample program, 265–267
capacity( ) function, 371
CascadingException sample program, 340–342
case sensitivity
general discussion, 2, 22
statements, 29
variables, 29
catch keyword, 337–338, 340, 342, 345
cerr object, 316–317
char (character) variables
char string variables versus, 40
character sets, 43
defined, 39, 44
naming conventions, 45
char string (character string) variables
char variables versus, 40
defined, 39
naming conventions, 45
char_16t (UTF-16 character) variables, 43
char_32t (UTF-32 character) variables, 43
CharDisplay sample program, 113
cin input device, 30, 316
class keyword, 176
class members, 262. See also static members
classes
abstract
concept of, 295–297
passing, 298
subclasses of, 298
accessing members of, 177
creating objects from, 176
defined, 171, 175, 215
format of, 176
member functions
accessing other members from, 182
calling, 180–182
defining, 177, 179–180
defining in classes, 185–186
defining separately from classes, 187–188
identifying current object, 183
overloading, 188–189
reasons for using, 177
naming conventions, 176
objects versus, 215
protected members
function of, 208–209
general discussion, 207
giving nonmember functions access to, 211–213
need for, 208
protecting internal state of classes, 210
using classes with limited interfaces, 211
scope resolution, 183–184
clear( ) function, 320, 323, 375
clog object, 316
close( ) function, 323
code obfuscation, 426–428
Code::Blocks environment
32-bit program, 26
creating projects, 19–21
defined, 10
downloading, 5
installing
for Macintosh, 15–19
for Ubuntu Linux, 13–15
for Windows, 11–14
Management window, 21–22
windowed programs, 11, 26
Command Line Tools (Mac), installing, 16–17
comments
creating while coding, 412
defined, 27
length of, 28
compare( ) function, 371
comparison operators, defined, 54
compilers, defined, 10
compiling, defined, 10
computer language (machine language), 9
concat( ) function, 370
Concatenate sample program
original version of, 115–117
wide character version of, 119–120
concrete classes, 296
concrete functions, 296
conditional clause, 75
conditional compilation, 163
const keyword, 40, 129–130, 143, 160–161, 165
constants
Boolean, 40
floating-point, 37
integer, 40
long integer, 40
variables declared as, 40–41
constexpr constant, 160–161
ConstructArray sample program, 218–219
ConstructDataMembers sample program, 236–237
ConstructingMembers sample program, 233–235
ConstructMembers sample program, 219–221
ConstructMembersWithInitializers sample program, 237–239
constructors
cannot be virtual, 288
combining, 230–231
constructing arrays, 244
constructing complex data members, 233–237
constructing constant data members, 239
constructing duplexes, 219–221
constructing members with initializers, 237–239
constructing multiple objects, 218–219
constructing single objects, 217–218
constructing subclasses, 278–279
copy
assignment operator versus, 305–306
automatic, 250–251
example of, 248–250
overview, 247
reasons for using, 248
shallow versus deep copies, 252–255
temporary objects, 256–258
default, 231–233, 235
defined, 216
defining with arguments, 225–227
delegating, 230–231
inheritance of by subclasses, 279
move, 258–260, 312–314
multiple inheritance, 356–357
open files for input and output, 317–318
overloading, 228–231
rules for order of construction
global objects, 241–243
local objects, 240
order of declaration, 243
overview, 239–240
static objects, 240–241
type conversion, 245
ConstructorsCallingEachOther sample program, 230–231
ConstructorWArg sample program, 226
ConstructorWDefaults sample program, 229–230
ConstructSeparateID sample program, 235–236
ConstructStatic sample program, 240–241
continue command, 82
ContinueDemo sample program, 82
Conversion sample program
building, 24–25
comments, 27–28
creating project, 19–21
declarations, 29
downloading and installing project files, 23–24
entering code, 21–23
executing, 25–26
expressions
defined, 30
storing results of, 30–31
framework of, 27
I/O statements, 30
statements, 28–29
cookies, 425–426
copy constructors (X::X(const X&))
automatic, 250–251
example of, 248–250
overview, 247
reasons for using, 248
shallow versus deep copies, 252–255
temporary objects, 256–258
CopyConstructor sample program, 248–250
cout output device, 30, 316
__cplusplus constant, 164
.CPP extension, 9
create( ) function, 413
cur flag, 330
CustomExceptionClass sample program, 342–345

- D -
data( ) function, 412
Data Execution Prevention (DEP), 404–405
__DATE__ constant, 164
Debian Linux, 14–15
dec flag, 327, 334
decimal (base 10) number system, 59–60
declarations
automatic, 45
defined, 29
function, 92
prototype, 99–100
decltype( ) keyword, 45, 46
decrement (--) unary operator
defined, 50–51
looping statements, 74–75
order of precedence, 48
prefix and postfix versions, 51, 74–75
DeepCopy sample program, 254–255
default keyword, 232
DefaultCopyConstructor sample program, 250–251
#define command
defined, 154
defining constants to be used throughout program, 157–158
defining function-like macros with arguments, 158–159
inline functions, 160
delete keyword, 133–134, 222, 224,
233, 253
DemoAssignmentOperator sample program, 306–311
DemoMoveOperator, 313–314
demotion, defined, 45
denial of service (DoS) attacks, 382–383
DEP (Data Execution Prevention), 404–405
DestructMembers sample program, 222–224
destructors
closing files for input and output, 319
declaring virtual, 288–289, 414–416
destructing subclasses, 279
reasons for using, 221
using, 221–224
digital certificates, 429
display( ) function, 344
DisplayMonths sample program, 147
DisplayString sample program, 114, 140
division (/) binary operator, 48
do . . . while loops, 73
DoS (denial of service) attacks, 382–383
dot (.) operator, 304
double (double precision) variables
  accuracy, 38, 41
defined, 39
  naming conventions, 45
range, 38, 41
size, 41
double slash (/) comment characters, 28

• E •
editors, defined, 10
#else command, 163
empty( ) function, 372, 375
encryption, 429–430
end( ) function, 375, 377
end flag, 330
#endif command, 163
endl value, 55–56, 329
environments, defined, 10
eof( ) function, 322, 323
equality (==) logical operator
  assignment operator versus, 54
defined, 54
list class, 375
string class, 371
#error command, 164–165
error messages. See also exception handling mechanism
  creating compiler errors, 164
  enabling all, 409–410
error: expected primary-expression before ‘>’ token, 25
file state, 320
illegal SQL statement, 385
overview, 24–25
sent to cerr, 317
ESP (stack pointer), 390–392, 396
exception( ) function, 324
exception handling mechanism
  example of, 337–338
  exception specification, 339
overview, 414
process for, 340–341
reasons for, 339
rethrowing exceptions up the chain, 345–346
throwing objects, 342–345
executable files (.exe), 10, 154
explicit keyword, 245
expressions
  calculation, 30
defined, 30
  mathematical operations, 48–49
  mixed-mode, 44–45
  storing results of, 30–31
extended names (signatures), 98
extern keyword, 366
extraction (>>) operator, 304, 315–316, 371

• F •
factorial( ) function, 337–339
FactorialException sample program, 337–338
factoring
  abstract classes
    concept of, 295–297
    passing, 298
    subclasses of, 298
culling common properties from similar classes, 294–295
defined, 291
direct inheritance versus, 292–294
fail( ) function, 320, 322
_FILE_ constant, 164, 344
FileCopy sample program, 326–327
fileFunc( ) function, 345–346
fill( ) function, 324, 328, 334
final keyword, 286–287
find( ) function, 372, 374
fixed flag, 327
flags( ) function, 324, 327–328
float (floating-point) variables
  accuracy, 38, 41
calculation speed, 37
counting, 37
defined, 36, 39
logical operations, 57–58
naming conventions, 45
overflow, 42
overview, 36–37
range, 38, 41
size, 41
FloatAverage sample program, 37–38
floating-point constants, 37
flow-control statements
branch statements, 69–71
looping statements
autoincrement/autodecrement feature, 74–75
avoiding infinite loops, 78
break command, 80–82
continue command, 82
general discussion, 71
for loops, 75–78
nesting, 82–84
range-based loops, 79
while loops, 72–73
overview, 69
switch statements, 84–85
flush( ) function, 324, 329
for each loops (range-based loops), 79, 111–112, 144
for loops
conditional clause, 75
empty sections, 76
examples of, 76–78
format of, 75
initialization clause, 75
ForDemo sample programs, 76–78
ForEachDemo sample program, 79
forward declarations, 212
friend keyword, 211–213
__func__ constant, 164, 344
function declarations, 92
functional programming
abstraction, 170–171
classification, 172
member functions, 178–179
FunctionDemo sample program
calling sumSequence( ), 92–93
defining displayExplanation( ), 92
defining sumSequence( ), 92
overview, 89–92
splitting loops into multiple functions, 93
functions. See also member functions arguments
default values for, 101–102
defined, 93
functions with, 95–97
functions with multiple, 97
optional nature of, 94
passing by reference, 102–103
passing by value, 102–103
defined, 89, 93
example of
calling sumSequence( ), 92–93
defining displayExplanation( ), 92
defining sumSequence( ), 92
overview, 89–92
splitting loops into multiple functions, 93
main( ), 97
name collision, 156–157
overloading names of, 98–99
passing objects to
calling functions with object pointers, 196–197
calling functions with object values, 195–196
calling functions with reference operator, 198–199
memory consumption and copying objects, 199
passing pointers to
as arguments, 127–128
by reference, 128–129
by value, 127, 129
prototype declarations, 99–100
return values, 93–94
simple, 94
variable storage types, 104
fuzzing the program, 418

G

-g switch, 427
gcc compiler
defined, 10
downloading, 5
installing
for Macintosh, 16–19
for Ubuntu Linux, 14–15
for Windows, 11–13
gcount( ) function, 324, 325, 327
get( ) function, 324, 325
ggetline( ) function, 324, 325, 332, 402–403
getString( ) function, 388, 390–394, 396–398, 401–404

global objects
defined, 216, 240
going out of scope, 222
rules for order of construction, 239, 241–243
good( ) function, 320, 324, 327
GraduateStudent sample program, 278–279
greater than (> ) logical operator, 54–55, 371
greater than or equal to (>= ) logical operator, 54–55

• H •

.h extension, 157
hacker-proof code
avoiding user input assumptions, 417–418
botnets, 382–383
buffer overflow
avoiding, 397–402
call stack, 389–393
example of, 387–389
general discussion, 386
hacking, 394–397
operating system helps, 405–406
string class, 402–404
checking for illegal input, 418–419
code obfuscation, 426–428
denial of service attacks, 382–383
development process, 421
digital certificates, 429
encryption, 429–430
exception handling, 418–419
fuzzing the program, 418
general discussion, 381
hackers’ motives, 381–382
program logs, 419–421
remote session management, 425–426
secure user authentication, 423–425
SQL injection
avoiding, 386
example of, 383–385
version control, 421–423
HAS_A relationships, 280
head pointers, 202, 265

heap memory
allocating arrays of objects off, 200
allocating objects off, 199–200
assignment operator, 306
classes that automatically allocate, 201
constructors and destructors, 217, 221–223, 227, 252–254
exception handling mechanism, 345–346
limited scope, 131–133
managing systematically, 413
overview, 130–131
solving scope issues using, 133–134
hex flag, 327, 334
hexadecimal (base 16) number system, 61–62
hexadecimal character (\xnn), 42–43

• I •

identity theft, 382
#if command, 154, 162–163
if statements, 70–71
#endif command, 163
ifndef command, 163
ifstream class, 317, 319, 320–321
in flag, 318, 319
#include command, 157
defined, 154
enclosing file in quotes versus brackets, 155
include files, 155, 157, 186
increment (++) unary operator
defined, 50–51
iterators, 377
looping statements, 74
order of precedence, 48
prefix and postfix versions, 51
reasons for creating, 51
indentation, 22, 71
inequality (!= ) logical operator, 54
infinite loops, 78
inheritance
defined, 171, 273–274
example of, 275–277
inheritance (continued)
factoring
abstract classes, 295–298
culling common properties from similar classes, 294–295
defined, 291
direct inheritance versus, 292–294
HAS_A relationships, 280
IS_A relationships, 274–275, 291
multiple inheritance mechanism
avoiding, 416
disadvantages of, 357
general discussion, 347
name collision, 349–351
object construction, 356–357
overview, 348–349
virtual inheritance, 350–356
need for, 274–275
polymorphism (late binding)
constructors, 288
declaring functions as not overrideable, 287
declaring functions virtual, 285–286
defined, 284
destructors, 288–289
need for, 284
overriding base class functions, 281–284, 286–287
static member functions, 287
subclasses
constructing, 278–279
destructing, 279
inheriting constructors, 279
overview, 277
transitive nature of, 275
InheritanceExample sample program, 275–277
initialization clause, 75
initializer lists
arrays, 110
defined, 244
invoking constructors other than default, 244
range-based loops, 79
templates, 366–368
inline functions, 160
advantages of, 186
defining member functions in classes, 185–186
inline keyword, 160
insert( ) function, 372, 374, 375
insertion (<<) operator, 304, 315–316, 371
instances, defined, 171, 176
int (integer) variables
accuracy, 41
defined, 34, 38
logical operations, 57
naming conventions, 45
overflow, 42
range, 36, 41
size, 41
truncation, 35–36
IntAverage sample program, 37, 154
integer constants, 40
integers, defined, 34
intrinsic constants, 164–165
I/O statements, 30
IS_A relationships, 274–275, 291
isLegal sample program, 101
istream class, 316, 323, 329–330, 371
istringstream class, 330, 332–333
istrstream class, 330
iteration, defined, 110
iterators, 376–378

• J •
JMP instruction, 396

• L •
late binding. See polymorphism
Layout sample program, 123–124
lcd( ) function, 414
left flag, 327
length( ) function, 372
less than (<) logical operator, 54–55, 371
less than or equal to (<=) logical operator, 54–55
level of abstraction, 170
__LINE__ constant, 164, 344
linked lists
adding objects to head of, 202
advantages of, 201
declaring linkable classes, 202
declared, 201
example of, 203–206
head pointers, 202
moving through elements in, 203
LinkedListClass sample program, 268
LinkedListData sample program, 203–206
linker, 154
linking, defined, 10
list class
example of, 378–380
iterators, 376–378
methods of, 375–376
operations on entire lists, 378
overview, 375
lists
initializer
arrays, 110
defined, 244
invoking constructors other than default, 244
range-based loops, 79
templates, 366–368
linked
adding objects to head of, 202
advantages of, 201
declaring linkable classes, 202
defined, 201
element of, 203–206
head pointers, 202
moving through elements in, 203
Little Endian, 391, 393
local objects
defined, 216
going out of scope, 222
rules for order of construction, 239–240
logical operations
bitwise logical operators
defined, 62
general discussion, 53
test of, 65–67
using, 64
values, 63–64
reasons for using, 53–54
short-circuit evaluation, 58–59
simple logical operators, 53, 54
storing logical values
Boolean variables, 55–56
floating-point variables, 57–58
integer variables, 57
long double variables
accuracy, 41
defined, 39
range, 41
size, 41
long int (long integer) variables
accuracy, 41
defined, 38
naming conventions, 45
range, 41
size, 41
long integer constants, 40
long long int (long long integer)
variables
accuracy, 41
defined, 39
range, 41
size, 41
looping statements
autoincrement/autodecrement feature, 74–75
avoiding infinite loops, 78
break command, 80–82
continue command, 82
general discussion, 71
for loops, 75–78
nesting, 82–84
range-based loops, 79
while loops, 72–73

\textbf{M}

machine language (computer language), 9
Macintosh
installing Code::Blocks environment, 17–19
installing Command Line Tools, 16–17
installing Xcode, 16
MacroConfusion sample program, 158
main( ) function
  accessing arguments to, 148–149
    from Code::Blocks, 150
    from command prompt, 149–150
    from Windows, 150–151
  overview, 97
  rules for order of construction, 241–242
manipulators, 333–335
mathematical operations
  applying to pointer variables, 135–136
  assignment operators, 51–52
  binary operators, 47–48
  expressions, 48–49
  general discussion, 47
  order of precedence, 49–50
  unary operators, 50–51
max_size( ) function, 372
maximum( ) function, 359–361
MaxTemplate sample program, 360–361
MD5 algorithm, 423–424
member functions
  accessing other members from, 182
  calling, 180–182
constructors
  cannot be virtual, 288
  combining, 230–231
  constructing arrays, 244
  constructing complex data members, 233–237
  constructing constant data members, 239
  constructing duplexes, 219–221
  constructing members with initializers, 237–239
  constructing multiple objects, 218–219
  constructing single objects, 217–218
copy, 247–258
default, 231–233, 235
defined, 216
defining with arguments, 225–227
delegating, 230–231
move, 258–260, 312–314
multiple inheritance, 356–357
open files for input and output, 317–318
overloading, 228–231
rules for order of construction, 239–243
type conversion, 245
defining, 177, 179–180
defining in classes, 185–186
defining separately from classes, 187–188
destructors
  declaring virtual, 288–289
  reasons for using, 221
  using, 221–224
identifying current object, 183
overloading, 188–189
overriding base class functions, 281–284
reasons for using, 177
static, 265–267, 287
virtual
  constructors, 288
  declaring, 285–286
  declaring functions as not overrideable, 287
destructors, 288–289
overriding base class functions, 286–287
pure, 296
static, 287
methods, 180. See also member functions
Microsoft Windows, installing Code::Blocks environment, 11–14
MinGW Compiler Suite, 12
mixed-mode expressions, 44–45
modules
  defined, 99
  headers, 410
modulus (%) binary operator
  defined, 48
  order of precedence, 48
  syntax changes, 304
move constructors (X::X(X&&)), 258–260, 312–314
MoveCopy sample program, 258–260
multiple inheritance mechanism
  avoiding, 416
  disadvantages of, 357
  general discussion, 347
  name collision, 349–351
  object construction, 356–357
  overview, 348–349
  virtual inheritance, 350–356
MultipleInheritance sample program, 348–349
MultipleInheritanceFactoring sample program, 351–354
multiplication (*) binary operator, 48, 137
Index

\n (newline character), 28, 42, 154, 319, 329
name collision, 156–157, 349–350
NameDataSet sample program, 268–269
namespaces, 156–157
negative (−) unary operator
defined, 50
order of precedence, 48
nested loops, 82–84
NestedDemo sample program, 83–84, 90
new keyword, 133–134
newline character (\n), 28, 42, 154, 319, 329
No Operation (NOP) instruction, 396
NoBufferOverflow sample programs,
398–403
nonmember functions
defined, 180
giving access to protected members,
211–213
NOP (No Operation) instruction, 396
NOT (!) logical operator, 54
NOT (~) bitwise operator
defined, 62
destructors, 221
using, 64
values, 63
NTBS (null-terminated byte strings),
114–115
null character (\0), 42–43, 114–115
nullptr constant, 144–145, 149, 202,
309, 418
null-terminated byte strings (NTBS),
114–115
number systems
binary (base 2), 60
decimal (base 10), 59–60
hexadecimal (base 16), 61–62
octal (base 8), 60–61
Nx flag, 404

O

object files (.o), 154
object-oriented programming (OOP)
abstraction, 169–170
classification, 171–172
functional programming versus, 170–172
objects
allocating off heap memory, 199–200
arrays of
allocating off heap memory, 200
declaring, 191–192
classes versus, 215
creating from classes, 176
defined, 215
global
defined, 216, 240
going out of scope, 222
rules for order of construction, 239,
241–243
local
defined, 216
going out of scope, 222
rules for order of construction, 239–240
naming conventions, 176
object pointers
arrow pointers, 194
declaring, 192–193
dereferencing, 193–194
passing to functions
calling functions with object pointers,
196–197
calling functions with object values,
195–196
calling functions with reference
operator, 198–199
memory consumption and copying
objects, 199
static
defined, 240
rules for order of construction, 239–241
ObjPtr sample program, 192–193
oct flag, 327, 334
octal (base 8) number system, 60–61
code (\0nn), 42–43
ofstream class, 317–318, 319
online resources
Budget sample program source code, 3
Cheat Sheet (companion to book), 4
Code::Blocks environment, 5
Command Line Tools (Mac), 16–17
Frequently Asked Questions (FAQ), 5
gcc compiler, 5
source code (companion to book), 4
Xcode development package, 16
OOP. See object-oriented programming
open( ) function, 324
openFile( ) function, 321–322
operators
  assignment, 30
defined, 30
  functions versus, 303–304
overloading, 304, 413
OR (|) bitwise operator
defined, 62
test program using, 65–67
values, 63
OR (||) logical operator
defined, 54–55
  short-circuit evaluation, 58
ostream class, 316, 323, 371
ostringstream class, 330, 333
ostringstream class, 330
out flag, 318, 319
outline functions, 187–188
OverloadConstructor sample program, 228–229
overloading
  assignment operators, 305, 306–311
constructors, 228–231
  functions, 98–99
member functions, 188–189
operators, 304, 413
subscript operator, 311
OverloadOverride sample program, 282–283, 285
override descriptor, 286–287

parameters, 148
parseString( ) function, 332–333
PassObjPtr sample program, 196–197
PassObjRef sample program, 198–199
PassObjVal sample program, 195–196
pointer variables
  accessing arguments to main( )
    from Code::Blocks, 150
    from command prompt, 149–150
overview, 148–149
    from Windows, 150–151
addresses, 122–123
applying operators to, 135–136, 142
arrays
  applying operators to addresses, 138–139
  of character strings, 146–147
  contrasted with pointers, 142–144
  of pointers, 145–146
relationship to pointers, 136–138
const keyword, 129–130
defined, 121
head pointers, 202
heap memory block
  limited scope, 131–133
overview, 130–131
  solving scope issues using, 133–134
link pointers, initializing, 202, 205
nullptr constant, 144–145
object pointers
  arrow pointers, 194
declaring, 192–193
dereferencing, 193–194
operators, 123–124
  passing to functions
    as arguments, 127–128
    by reference, 128–129
    by value, 127, 129
  positioning within files, 329–330
string manipulation
  with character pointers, 139–141
  improving efficiency with, 141
using, 125–126
variable size, 121–122
polymorphism (late binding)
constructors, 288
declaring functions as not overrideable, 287
declaring functions virtual, 285–286
defined, 284
destructors, 288–289
need for, 284
overriding base class functions, 281–284, 286–287
static member functions, 287
pop_back( ) function, 375
pop_front( ) function, 375
positive (+) unary operator
  defined, 50
order of precedence, 48
precision( ) function, 324, 334
preprocessor commands
  #define command, 154, 157–162
  #if command, 154, 162–163
  #include command, 154–155, 157
defined, 153–154
intrinsic constants, 164–165
typedef keyword, 166
PrintArgs sample program, 148–151
private keyword, 209, 411
program logs (system logs)
  maintaining, 420–421
  overview, 419–421
  reviewing, 420
projects
  creating, 19–21
  defined, 19
  naming, 21
promotion, defined, 45
protected keyword. See protected members
protected members
  declaring constructors as, 257
  function of, 208–209
  general discussion, 207
  giving nonmember functions access to, 211–213
  limiting visibility of class internals, 411
need for, 208
  protecting internal state of classes, 210
  using classes with limited interfaces, 211
public keyword, 176, 208, 277
push_back( ) function, 376
push_front( ) function, 376
put( ) function, 324, 325

• R •
range-based loops (for each loops), 79, 111–112, 144
rbegin( ) function, 376
read( ) function, 324, 325–327
Red Hat Linux, 14–15
release( ) function, 413
remove( ) function, 376
removeSpaces( ) function, 374
rend( ) function, 376
replace( ) function, 372
resize( ) function, 372
resolving calls, 189
return 0 statement, 31
right flag, 327

• S •
salt value, 423–424
SavingsClassOutline sample program, 187–188
scientific flag, 327
scope
  defined, 131
  limited, 131–134
scope resolution (::) operator, 183–184, 304, 375
secure hash, 423–424
Secure Socket Layer (SSL), 426
security
  avoiding user input assumptions, 417–418
  checking for illegal input, 418–419
code obfuscation, 426–428
development process, 421
digital certificates, 429
encryption, 429–430
exception handling mechanism
  example of, 337–338
  exception specification, 339
overview, 418–419
process for, 340–341
reasons for using, 339
rethrowing exceptions up the chain, 345–346
throwing objects, 342–345
fuzzing the program, 418
hacker-proof code
  botnets, 382–383
  buffer overflow, 386–406
denial of service attacks, 382–383
general discussion, 381
hackers’ motives, 381–382
SQL injection, 383–385, 386
security (continued)
multiple inheritance mechanism
  avoiding, 416
  disadvantages of, 357
  general discussion, 347
  name collision, 349–351
  object construction, 356–357
  overview, 348–349
  virtual inheritance, 350–356
operators
  functions versus, 303–304
  insertion and extraction, 304
  move constructor and operator, 312–314
  overloading assignment operator, 306–311
  overloading subscript operator, 311
  shallow copies, 305–306
program logs, 419–421
remote session management, 425–426
secure user authentication, 423–425
Standard Template Library
  general discussion, 369–370
  list class, 375–380
  string class, 370–375
stream I/O
  default stream objects, 316–317
  endl object, 329
  example of, 320–321
  file state, 320
  format control, 327–329
  function of, 315–316
  manipulators, 333–335
  open file modes, 319
  opening and closing files, 317–319
  positioning pointers within files, 329–330
  reading and writing directly to streams, 325–327
  reading from files, 319
  stringstream subclasses, 330–333
  writing to files, 319
templates
  class template templates, 362–365
  code generation, 365
  compilation and error checking, 366
  external instantiations, 366
  general discussion, 359–360
  generalizing functions into, 360–361
  initializer lists, 366–368
  instantiating, 361
  memory consumption, 366
  string class, 374–375
  version control, 421–423
seekg( ) function, 324, 330
seekp( ) function, 325
segment violation errors, 143
semicolons (;), 28
set manipulator, 334
setf( ) function, 325, 327
setw( ) function, 334
SHA256 algorithm, 423–424
ShallowCopy sample program, 252–253
short int (short integer) variables
  accuracy, 41
  defined, 38
  range, 41
  size, 41
short-circuit evaluation, 58–59
showbase flag, 327
showpoint flag, 328
signatures (extended names), 98
simple logical operators, 53, 54
size( ) function, 372, 376
sizeof keyword, 121
skipws flag, 328
sort( ) function, 376, 378, 380
source files, defined, 9
splice( ) function, 376
SQL (Structured Query Language) injection
  avoiding, 386
  checking for illegal input, 418–419
  example of, 383–385
SquareDemo sample program, 95–97
SSL (Secure Socket Layer), 426
stack pointer (ESP), 390–392, 396
Standard Template Library (STL)
general discussion, 369–370
list class
  example of, 378–380
  iterators, 376–378
  methods of, 375–376
  operations on entire lists, 378
  overview, 375
Index

string class
   capabilities of, 373–374
   methods of, 370–372
   overview, 370
   template, 374–375
statements
   branch, 69–71
   case sensitivity, 29
   declaration, 29
   defined, 28
   expressions, 30–31
I/O, 30
looping
   autoincrement/autodecrement feature, 74–75
   avoiding infinite loops, 78
   break command, 80–82
   continue command, 82
   general discussion, 71
   for loops, 75–78
   nesting, 82–84
   range-based loops, 79
   while loops, 72–73
switch, 84–85, 146
whitespace, 28
static keyword, 104, 261
static members
   defined, 261
   static data members
      counting objects, 264–265
      flagging actions, 265
      need for, 261–262
      providing space for head pointer, 265
      referencing, 263–264
      using, 262–263
   static member functions, 265–267, 287
   this keyword, 268–269
static objects
   defined, 240
      rules for order of construction, 239–241
static_assert( ) function, 165
_STDCC_constant, 164
STL. See Standard Template Library
STLListStudents sample program, 378–380
STLString sample program, 373–374
strcat( ) function, 118
strncpy( ) function, 118
strncpy( ) function, 118
strncpy( ) function, 118
strncpy( ) function, 118
strncpy( ) function, 118
strstr( ) function, 118
struct keyword, 176
Structured Query Language. See SQL injection
subclasses
   of abstract classes, 298
   constructing, 278–279
   defined, 171
   destructing, 279
   inheriting constructors, 279
   overview, 277
submit( ) function, 384
subscript operator, overloading, 311
strcpy( ) function, 118, 143, 359
stream I/O
   default stream objects, 316–317
   endl object, 329
   example of, 320–321
   file state, 320
   format control, 327–329
   function of, 315–316
   manipulators, 333–335
   open file modes, 319
   opening and closing files, 317–319
   positioning pointers within files, 329–330
   reading and writing directly to streams, 325–327
   reading from files, 319
   stringstream subclasses, 330–333
   writing to files, 319
StreamInput sample program, 320–323
StreamOutput sample program, 318
string class
   capabilities of, 373–374
   methods of, 370–372
   overview, 370
   template, 374–375
string manipulation
   with character pointers, 139–141
   functions for, 117–118
   wide functions for, 118–120
string variable, 120
StringStream sample program, 330–333
substr( ) function, 372
subtraction (-) binary operator, 48, 136
switch statements, 84–85, 146
system logs. See program logs
sz prefix, 115

T
\t (tab character), 42–43
tellg( ) function, 325, 329
tellp( ) function, 325, 330
template keyword, 361
Template sample program, 27
templates
class template templates, 362–365
code generation, 365
compilation and error checking, 366
external instantiations, 366
general discussion, 359–360
generalizing functions into, 360–361
initializer lists, 366–368
instantiating, 361
memory consumption, 366
Standard Template Library
general discussion, 369–370
list class, 375–380
string class, 370–375
stream I/O, 317
string class, 374–375
TemplateVector sample program, 362–365, 367–368
ternary (?:) operator, 304
this keyword, 268–269
throw keyword, 337–338, 342–346
__TIME__ constant, 164
__TIMESTAMP__ constant, 164
Transport Layer Security (TLS), 426
trunc flag, 318, 319
truncation
avoiding with floating-point variables, 36–37
defined, 35
logical operations, 57–58
overview, 35–36
try keyword, 337–338, 340, 342
typedef keyword, 166

U
Ubuntu Linux
installing Code::Blocks environment, 15
installing gcc compiler, 14–15
unary operators, order of precedence, 48
Unicode Transformation Format (UTF), 43–44
unique( ) function, 376
unitbuf flag, 328
unsetf( ) function, 325, 327
unsigned keyword, 39
unsigned variables, 39
unwinding the stack, 340
uppercase flag, 328
user authentication, 423–425
UTF (Unicode Transformation Format), 43–44
UTF-16 character (char_16t) variables, 43
UTF-32 character (char_32t) variables, 43

V
variables. See also pointer variables
automatic, 110
Boolean
defined, 39, 44
list class, 375
logical operations, 55–56
string class, 371–372
character
char string variables versus, 40
character sets, 43
defined, 39, 44
naming conventions, 45
character string
char variables versus, 40
defined, 39
naming conventions, 45
common types, 38–40
declaring
automatically, 45
constant, 40–41
overview, 33–34
defined, 29, 33
double precision
  accuracy, 38, 41
defined, 39
naming conventions, 45
range, 38, 41
size, 41
floating-point
  accuracy, 38, 41
calculation speed, 37
counting, 37
defined, 36, 39
logical operations, 57–58
naming conventions, 45
overflow, 42
overview, 36–37
range, 38, 41
size, 41
integer
  accuracy, 41
defined, 34, 38
logical operations, 57
naming conventions, 45
overflow, 42
range, 36, 41
size, 41
truncation, 35–36
long double
  accuracy, 41
defined, 39
range, 41
size, 41
long double precision
  accuracy, 41
defined, 39
range, 41
size, 41
long integer
  accuracy, 41
defined, 38
naming conventions, 45
range, 41
size, 41
long long integer
  accuracy, 41
defined, 39
range, 41
size, 41
mixed-mode expressions, 44–45
naming, 29
nonprintable characters, 42–43
overflow, 42
range, 41
short integer
  accuracy, 41
defined, 38
range, 41
size, 41
signed and unsigned versions, 39
syntax, 29
unsigned, 39
UTF-16 character, 43
UTF-32 character, 43
wide character, 39, 43–44
VariableSize sample program, 121–122
vectors, 362–365
version control, 421–423
virtual inheritance, 350–356
virtual keyword, 285, 354–355
virtual member functions
  constructors, 288
declaring, 285–286
declaring functions as not
    overrideable, 287
destructors, 288–289
overriding base class functions, 286–287
pure, 296
static, 287
VirtualInheritance sample program,
    354–356
Visual Studio, 5
Visual Studio Express, 5
void keyword, 94, 372, 375–376

W

warning messages, enabling all, 409–410
wcerr object, 316–317
wchar_t (wide character) variables, 39, 43–44
wcin object, 316–317
wclog object, 316
wcout object, 316–317
wcscat( ) function, 119
wcscmp( ) function, 119
wcscpy() function, 119
wcslen() function, 119
wcsncat() function, 119
wcsncmp() function, 119
wcsncpy() function, 119
wcsstr() function, 119
WEP (Wired Equivalent Privacy), 430
while loops
  autodecrement feature, 74–75
  overview, 72–73
WhileDemo sample program, 72–74
whitespace, 2
  defined, 28
  enforcing, 71
wide character (wchar_t) variables, 39, 43–44
wide string-handling functions, 118–120
width() function, 325, 328–329
Wi-Fi Alliance, 430
Wi-Fi Protected Access (WPA1 and WPA2) standards, 430
wfstream class, 317
Wired Equivalent Privacy (WEP), 430
wifstream class, 317
workspaces, defined, 24
wostream class, 316
WPA1 and WPA2 (Wi-Fi Protected Access) standards, 430
write() function, 325–327
wstring class, 374

• X •
Xcode development package, installing, 16
\xnn (hexidecimal character), 42–43
XOR (^) bitwise operator
  defined, 62
  test program using, 65–67
  values, 64
XOR instruction, 397

• Z •
zombies, 383