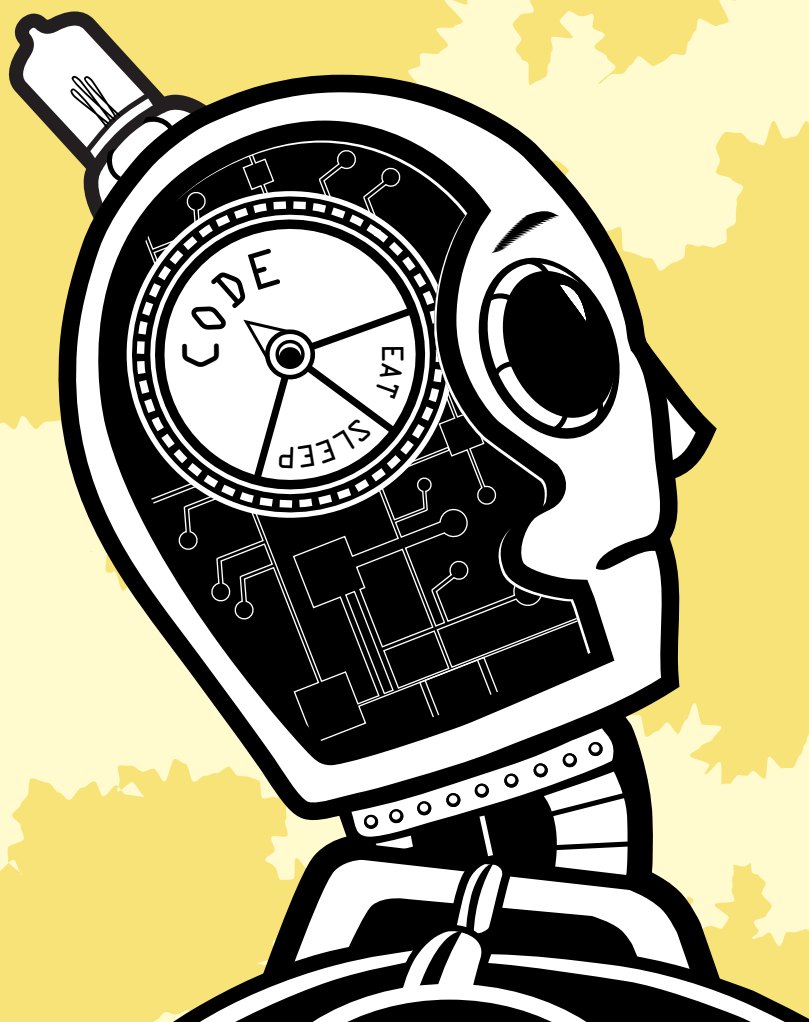


THINK LIKE A PROGRAMMER

AN INTRODUCTION TO
CREATIVE PROBLEM SOLVING

V. ANTON SPRAUL



INDEX

Numbers and Symbols

- && operator (logical *and*), 48
 - short-circuit evaluation of, 129, 132, 133
- & operator (address-of), 85
- & symbol (reference parameter), 84–85, 137, 211, 213
- * operator (dereference), 59–60, 82, 128–129, 138, 213–216
- * symbol (pointer declaration), 59, 75, 82, 85, 99–100, 160, 177–178, 186, 192
 - pointer to function, 177–178
- == operator (equality), 197–198
- = operator (assignment), 137–138, 197–198
- > operator (structure deference), 102, 128
- % operator (modulo), 33–34, 39–40, 50–52

A

- abstract data type, 116, 175, 183, 188–189
- access specifier, 112, 119, 125, 127
- activation record, 86–87, 89–90
- address-of operator (&), 85
- algorithm, xv, 173–174, 176–177, 182–183, 188–193
- analogy. *See* finding an analogy
- and* (Boolean logic), 48
 - short-circuit evaluation of, 129, 132, 133
- application programming interface (API), 176

- arrays, 56
 - ARRAY_SIZE constant, 58
 - aggregate statistics, 61–62
 - basic operations, 56–62
 - of bool, 209, 215
 - computing average, 61
 - const array declaration, 67
 - copying, 57
 - dynamically allocating, 93, 97, 98
 - element, 56
 - finding largest value in, 58–59, 66, 70–71, 73
 - of fixed data, 67–69
 - initialization, 57, 70, 71
 - median, 67
 - mode (statistic), 62–65
 - multidimensional, 71–74
 - treating as array of arrays, 72–74
 - when to use, 71–72
 - nonscalar, 69–71
 - recursive processing of, 153–155
 - searching
 - criterion-based, 58–59
 - for specific value, 58
 - sorting, 59–61, 189–193
 - insertion sort, 60–61, 190–192, 193
 - qsort, 59–60, 192–193
 - of string, 123
 - of struct, 69–71
 - subscript, 56, 66
 - vs. vectors, 75–76
 - when to use, 74–78
- assignment operator (=), 137–138, 197–198

avoiding frustration, 21–22, 95–96,
201, 220, 224
by dividing problems, 41

B

bad_alloc exception, 89
bad smells, 65, 97, 192
base case, 144, 162
Big Recursive Idea (BRI), 143,
152–155
binary tree
empty, testing for, 162
leaf, 163
recursive processing, 160–165,
166–167
root node, 161
subtree, 161

C

C++
array declaration, 55
array initialization, 57
as choice for this book, xvii
cin standard stream, 26
class declaration, 112–113
cout standard stream, 26
delete operator, 83
exception, 130
file processing, 210–211
free function, 88
friend keyword, 184
get method, 34
header files for input/output, 26
list class, 182–183, 210–214,
216, 218
malloc function, 88
new operator, 75, 82, 97, 98
pointer declaration, 82
prerequisites, xv
reference parameters, 84
short-circuit evaluation, 129,
132, 133
Standard Template Library, 175
this keyword, 120
typedef keyword, 91, 101, 127,
160, 177
character codes, 34–35

checksum validation, 31–32
cin standard stream, 26
class
access specifier, 112, 119,
125, 127
basic framework, 119–122
composition, 126
constructor, 112–113, 119,
121–122, 126–127
data member, 112
declaration, 112–113
deep copy, 134–137
destructor, 133–134
dynamic data structures,
125–140
encapsulation, 114, 126, 180
expressiveness, 117–118, 121, 128
fake, 140–141
friend method, 184
get and *set*, 119–121
goals of use, 113–118
information hiding, 115, 180
interface, 115
method, 112
method names, choosing, 117,
119–120
operator overloading, 137
private member, 112
protected member, 112
public member, 112
shallow copy, 135
single-tasker, 141
subclass, 112
support method, 122
template, 141
validation, 121, 124
wrapper function, 163–165
classic puzzles
the Fox, the Goose, and the
Corn, 3–7, 15, 17, 20
sliding number puzzle, 7–11, 18
sudoku, 11–13
Quarrasi Lock, 13–15, 20
code block, 173
code reuse, 53, 172–173
abstract data type, 175
algorithm, 173–174
as-needed learning, 180–188

- class use, 114
 - code block, 173
 - component, 173
 - choosing, 188–193
 - finding, 182–183
 - exploratory learning, 176–180
 - library, 175–176
 - pattern, 174
 - properties, desired, 172
 - saving code for later use, 44, 67, 218
 - code validation. *See* testing
 - comparator function, 59
 - component, 173
 - types, 173–176
 - flexibility of, 188–189
 - composition, 126
 - const
 - arrays, 67–69, 71
 - numeric types, 58
 - parameters, 59, 211
 - constraints, 1–2, 6, 11–13, 19, 31, 33, 38, 40–41, 203
 - importance of, 26
 - constructor, 112–113, 119, 121–122, 126–127
 - copy constructor, 138
 - default constructor, 113, 122, 179
 - converting between ranges
 - character digit to integer, 35, 43–48
 - number to letter of alphabet, 49
 - copy-and-paste job, 173
 - copy constructor, 138
 - cout standard stream, 26
 - creeping featurism, 201
 - cross-linking, 100, 103, 134–135
 - cross-training, 220
 - c_str method, 211
- D**
- dangling reference, 90, 100, 125, 212
 - caused by cross-linking, 136
 - data member, 112, 119–120
 - data redundancy, 123–124
 - deep copy, 134–137
 - default constructor, 113, 122, 179
 - dereferencing, 82
 - design pattern. *See* pattern
 - destructor, 133–134
 - diagrams, pointer, 92, 94, 96, 103
 - direct recursion, 144
 - DirectX, 176
 - dispatcher function, 153–154
 - dividing problems, 17–18, 31–41, 41–53
 - class use, 115
 - sliding tile puzzle, 8–11
 - division by zero, 108, 198
 - doubly linked list, 131
 - dummy record, 129, 179, 181, 186
 - dynamic data structures, 158–165
- E**
- efficiency, 181–182, 193
 - encapsulation, 114, 126, 180
 - end-of-line
 - character code for, 37
 - finding in character stream, 38
 - equality operator (==), 197–198
 - exception, 130
 - experimenting with programs, 20–21, 28, 30, 37
 - expressiveness, 117–118
- F**
- fake class, 140–141
 - fast learner, 200–201
 - fast coder, 200–201
 - fencepost error, 196
 - file processing, 210–211
 - finding an analogy, 2, 20, 62, 93, 182, 191
 - creating your own analogy, 38–39
 - loop problems, 29–30
 - Quarrasi Lock problem, 13–15
 - find method (string), 211–212
 - flexibility, 93, 154, 160, 188–189
 - the Fox, the Goose, and the Corn, 3–7, 15, 17, 20

functions
 activation record, 86
 comparator, 59
 dispatcher, 153–154
 multiple exits, 132
 names, choosing, 117, 119–120
 pointer to, 177
 recursive, 152–165
 wrapper, 163–165
frustration, 21. *See also* avoiding
 frustration

G

get method (general), 119
get method (*iostream*), 34

H

hangman, 204–218
head pointer, 103, 123, 127, 137
head recursion, 144, 146–147,
 151–152
heap, 87–88
 overflow, 89
helper function, 98
histogram, 65–66

I

indirect recursion, 144
inefficiency
 in space, 77
 in time, 77, 181–182
information hiding, 115–117
input processing, 31–41
iteration, 25. *See also* looping
iterator class, 183, 210
 begin method, 183
 const_iterator, 211
 end method, 183
 erase method, 212
 find method, 211–212
iterator pattern, 183–187
 advancing to next node, 185
 benefits, 183
 initializing, 185
 methods, 184

J

Java, xiv, 111, 176, 221
JDBC, 176

K

King of the Hill algorithm, 58, 66,
 70–71, 73, 214–215
Kobayashi Maru, 2, 19, 26

L

learning new skills, 219–224
 classwork, 223–224
 for known languages, 222
 libraries, 223
 new languages, 219–222
left-hand side, 137
library, 175–176, 223
lifetime, 90
linked lists, 101–108, 175
 adding node to, 104–106, 128
 building, 101–103
 diagram, 103
 doubly linked list, 131
 empty, testing for, 108
 head pointer, 103, 123, 127, 137
 iterator, 182–187
 node, 101, 127
 NULL terminator, 103
 recursion, 168–169
 recursive processing, 158–160
 removing node, 130–133
 reverse traversal, 168–169
 sequential access, 103
 traversal, 106–108, 129, 168–169,
 179, 181
list class, 182–183, 210–214,
 216, 218
lookup table, 67
looping, 26–41, 71, 94
loop postmortem, 217

M

master plan, 196–203
median, 67
member, 112

- memory allocation
 - activation record, 86
 - array, 74, 97
 - bad_alloc exception, 89
 - in classes, 125–140
 - dangling reference, 90, 100
 - delete operator, 83
 - fragmentation, 87–88
 - free function, 88
 - heap, 87–88
 - heap overflow, 89
 - leak (*see* memory leak)
 - lifetime, 90
 - malloc function, 88
 - new operator, 75, 82, 97, 98
 - reasons to minimize, 88–90
 - stack, 86–87, 89–90
 - thrashing, 89
 - memory fragmentation, 87–88
 - memory leak, 75, 90
 - avoiding, 95
 - minimal data set, 160
 - mode (statistic), 62
 - modulo operator (%), 33–34, 37, 39–40, 50–52
 - most constrained variable, 12
 - multidimensional array, 71–74
 - treating as array of arrays, 72–74
 - when to use, 71–72
- N**
- new operator, 75, 82, 97, 98
 - node
 - binary tree, 160–161, 163
 - linked list, 101, 127
 - payload, 102, 145
 - npos value, 211–212
 - NULL pointer, 90
- O**
- OpenGL, 223
 - operators
 - address-of (&), 85
 - assignment (=), 137–138, 197–198
 - dereference (*), 59–60, 82, 128–129, 138, 213–216
 - equality (==), 197–198
 - logical *and* (&&), 48
 - short-circuit evaluation of, 129, 132, 133
 - modulo (%), 33–34, 39–40, 50–52
 - overloading, 137–138
 - overconfidence, 199
 - overflow
 - heap, 89
 - stack, 89–90
 - overloading, 137–138
- P**
- parameters
 - recursive functions, use in, 155–156
 - reference, 84
 - pattern, 174
 - iterator, 183–187
 - policy, 176–180
 - singleton, 174
 - strategy, 176–180
 - wrapper function, 174
 - performance
 - inefficiency in space, 77, 85
 - inefficiency in time, 77, 181–182, 193
 - tuning, 77
 - planning, 16–17, 33, 95–96, 173
 - individuality of, 40
 - master plan, 196–203
 - pointers
 - benefits of, 83–84
 - cross-linking, 100
 - declaration, 59, 75, 82, 85, 99–100, 160, 177–178, 186, 192
 - dereferencing, 59–60, 82, 128–129, 138, 213–216
 - diagrams, 92, 94, 96, 103
 - to function, 177
 - NULL pointer, 90
 - reference parameters, 84
 - when to use, 84
 - policy, 176–180
 - public member, 112
 - push_back method, 76
 - private member, 112

- problem solving, xiii–xv, 2, 203–219
- protected member, 112
- prerequisites, xv
- property (C#), 120
- pseudocode, 63
 - conversion to documentation, 64
 - solving problems with, 63–64

Q

- qsort, 59–60, 65, 192–193
 - comparator function, 59, 192
- Quarrasi Lock problem, 13–15, 20

R

- random access, 56, 78
- rapid prototyping, 201
- readability, 117
- recursion, 143
 - base case, 144
 - Big Recursive Idea, 143, 152–155
 - binary tree, 160–165
 - breadcrumb trail, 166–169
 - common mistakes, 155–158
 - direct, 144
 - dynamic data structures,
 - applying to, 158–165
 - head, 144, 146–147, 151–152
 - indirect, 144
 - linked list, 158–160
 - vs. stack, 166–169
 - tail, 144, 145–146, 149–150
 - when to use, 165–169
 - wrapper function, 163–165
- reducing problems, 19–20, 41–53, 63, 190
 - loop problems, 26–29
- redundant data, 123–124
- refactoring, 65–67, 180, 200
- reference parameters, 84–85, 137, 211, 213
 - const, 211, 213
- resizable data structure, 83
- restating problems, 17, 33, 42, 182, 193
 - the Fox, the Goose, and the Corn, 5–7
 - loop problems, 31

- restore point, 218
- reuse. *See* code reuse
- right-hand side, 137
- robust programs, definition of, 96
- root node, 161
- runtime-sized data structure, 83

S

- scalar variable, 55
- sequential access, 103
- sequential search, 58
- set* method, 119
- shallow copy, 135
- short-circuit evaluation, 129
- single-tasker, 141
- singleton, 174
- sliding number puzzle, 7–11, 18
- solving by sample case, 92–96
- sorting, 59, 176–177, 189–193
 - insertion sort, 60–61, 190–192, 193
 - qsort, 59–60, 192–193
- special cases, 96
 - checking for, 96–97, 100, 124, 128, 132, 198–199
- stack, 86
 - linked list, 175
 - overflow, 89–90
 - runtime, 86–87
- starting with what you know, 18–19, 62, 92
 - loop problems, 29–30
 - most constrained variable, 12
 - sudoku, 11–13
- strategy, 176–180
- string class, 119
 - array, 123
 - c_str* method, 211
 - find* method, 211–212
 - npos* value, 211–212
- strings, 91
 - array implementation, 91–100
 - copying, 98
 - C-style, 178
 - linked list implementation, 101–107
 - terminator, 93

- struct, 69
- structure deference (->), 102, 128
- subclass, 122
- subscript, 56
- sudoku, 11–13
- support method, 122–125

T

- tail recursion, 144, 145–146, 149–150
- template class, 141
- test-driven development, 200
- testing, 124, 190, 199–200, 215
 - memory leaks, 95
 - promoting ease of, 34, 57, 66, 70, 218
 - storing test programs, 44
 - test cases, coding, 93, 98–100, 130–134, 186–187
- this keyword, 120
- thrashing, 89
- tracking state, 50–51
- traversal, linked list, 106–108, 129, 168–169, 179, 181
- typedef keyword, 91, 101–102, 127, 160, 177

V

- validation
 - checksum 31–32
 - code (*see* testing)
 - data, 61–62, 92, 96, 121, 124–125
- vectors, 55
 - vs. arrays, 75–76
 - declaring, 76
 - push_back method, 76

W

- weaknesses
 - coding weaknesses, 196, 197–199
 - design weaknesses, 196, 199–200
- whitespace, 34
- wrapper function, 163–165, 174