

# **C++ Hacker's Guide**

*by Steve Oualline*

Copyright 2008, Steve Oualline. This work is licensed under the Creative Commons License which appears in Appendix F. You are free:

- to Share — to copy, distribute, display, and perform the work
- to Remix — to make derivative works

Under the following conditions:

- Attribution: You must attribute the work by identifying those portions of the book you use as “Used by permission of Steve Oualline (<http://www.oualline.com>) under the the Creative Commons License.” (The attribution should not in any way that suggests that Steve Oualline endorses you or your use of the work).
- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to the web page: <http://creativecommons.org/licenses/by/3.0/us/>.
- Any of the above conditions can be waived if you get permission from Steve Oualline.
- Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.

## Table of Contents

Real World Hacks.....	9
Hack 1: Make Code Disappear.....	10
Hack 2: Let Someone Else Write It.....	12
Hack 3: Use the const Keyword Frequently For Maximum Protection.....	12
Hack 4: Turn large parameter lists into structures.....	14
Hack 5: Defining Bits.....	16
Hack 6: Use Bit fields Carefully.....	18
Hack 7: Documenting bitmapped variables.....	19
Hack 8: Creating a class which can not be copied.....	21
Hack 9: Creating Self-registering Classes.....	22
Hack 10: Decouple the Interface and the Implementation.....	25
Hack 11: Learning From The Linux Kernel List Functions.....	27
Hack 12: Eliminate Side Effects.....	29
Hack 13: Don't Put Assignment Statements Inside Any Other Statements.....	30
Hack 14: Use const Instead of #define When Possible.....	31
Hack 15: If You Must Use #define Put Parenthesis Around The Value.....	32
Hack 16: Use inline Functions Instead of Parameterized Macros Whenever Possible.....	33
Hack 17: If You Must Use Parameterized Macros Put Parenthesis Around The arguments.....	34
Hack 18: Don't Write Ambiguous Code.....	34
Hack 19: Don't Be Clever With the Precedence Rules.....	35
Hack 20: Include Your Own Header File.....	36
Hack 21: Synchronize Header and Code File Names.....	37
Hack 22: Never Trust User Input.....	38
Hack 23: Don't use gets.....	40
Hack 24: Flush Debugging.....	41
Hack 25: Protect array accesses with assert.....	42
Hack 26: Use a Template to Create Safe Arrays.....	45
Hack 27: When Doing Nothing, Be Obvious About It.....	46
Hack 28: End Every Case with break or /* Fall Through */.....	47
Hack 29: A Simple assert Statements For Impossible Conditions.....	47
Hack 30: Always Check for The Impossible Cases In switches.....	48
Hack 31: Create Opaque Types (Handles) Which can be Checked at Compile Time.....	49
Hack 32: Using sizeof When Zeroing Out Arrays.....	51
Hack 33: Use sizeof(var) Instead of sizeof(type) in memset Calls.....	51
Hack 34: Zero Out Pointers to Avoid Reuse.....	53
Hack 35: Use strncpy Instead of strcpy To Avoid Buffer Overflows.....	54
Hack 36: Use strncpy instead of strcpy for safety.....	55

Hack 37: Use snprintf To Create Strings.....	56
Hack 38: Don't Design in Artificial Limits.....	57
Hack 39: Always Check for Self Assignment.....	58
Hack 40: Use Sentinels to Protect the Integrity of Your Classes.....	60
Hack 41: Solve Memory Problems with valgrind.....	61
Hack 42: Finding Uninitialized Variables.....	63
Hack 29: Valgrind Pronunciation.....	65
Hack 43: Locating Pointer problems ElectricFence.....	65
Hack 44: Dealing with Complex Function and Pointer Declarations.....	65
Hack 45: Create Text Files Instead of Binary Ones Whenever Feasible.....	67
Hack 46: Use Magic Strings to Identify File Types.....	69
Hack 47: Use Magic Numbers for Binary Files.....	69
Hack 48: Automatic Byte Ordering Through Magic Numbers.....	70
Hack 49: Writing Portable Binary Files.....	71
Hack 50: Make You Binary Files Extensible.....	72
Hack 51: Use magic numbers to protect binary file records.....	74
Hack 52: Know When to Use _exit.....	76
Hack 53: Mark temporary debugging messages with a special set of characters.....	78
Hack 54: Use the Editor to Analyze Log Output.....	78
Hack 55: Flexible Logging.....	79
Hack 56: Turn Debugging On and Off With a Signal.....	81
Hack 57: Use a Signal File to Turn On and Off Debugging.....	82
Hack 58: Starting the Debugger Automatically Upon Error.....	82
Hack 59: Making assert Failures Start the Debugger.....	88
Hack 60: Stopping the Program at the Right Place.....	90
Hack 61: Creating Headings within Comment.....	92
Hack 62: Emphasizing words within a paragraph.....	93
Hack 63: Putting Drawings In Comments.....	93
Hack 64: Providing User Documentation.....	94
Hack 65: Documenting the API.....	96
Hack 66: Use the Linux Cross Reference to Navigate Large Coding Projects.	99
Hack 67: Using the Pre-processor to Generate Name Lists.....	103
Hack 68: Creating Word Lists Automatically.....	104
Hack 69: Preventing Double Inclusion of Header Files.....	105
Hack 70: Enclose Multiple Line Macros In do/while.....	105
Hack 71: Use #if 0 to Remove Code.....	107
Hack 72: Use #ifndef QQQ to Identify Temporary Code.....	107
Hack 73: Use #ifdef on the Function Not on the Function Call to Eliminate Excess #ifdefs.....	108
Hack 74: Create Code to Help Eliminate #ifdef Statements From Function Bodies.....	109
Hack 75: Don't Use any "Well Known" Speedups Without Verification.....	112
Hack 76: Use gmake -j to speed up compilation on dual processor machines	

.....	115
Hack 77: Avoid Recompiling by Using ccache.....	117
Hack 78: Using ccache Without Changing All Your Makefiles.....	118
Hack 79: Distribute the Workload With distcc.....	119
Hack 80: Don't Optimize Unless You Really Need to .....	120
Hack 81: Use the Profiler to Locate Places to Optimize .....	120
Hack 82: Avoid the Formatted Output Functions.....	122
Hack 83: Use ++x Instead of x++ Because It's Faster.....	123
Hack 84: Optimize I/O by Using the C I/O API Instead of the C++ One.....	124
Hack 85: Use a Local Cache to Avoid Recomputing the Same Result.....	126
Hack 86: Use a Custom new/delete to Speed Dynamic Storage Allocation....	128
Anti-Hack 87: Creating a Customized new / delete Unnecessarily.....	129
Anti-Hack 88: Using shift to multiple or divide by powers of 2.....	130
Hack 89: Use static inline Instead of inline To Save Space.....	131
Hack 90: Use double Instead of Float Faster Operations When You Don't Have A Floating Point Processor.....	132
Hack 91: Tell the Compiler to Break the Standard and Force it To Treat float as float When Doing Arithmetic.....	133
Hack 92: Fixed point arithmetic.....	134
Hack 93: Verify Optimized Code Against the Unoptimized Version.....	138
Case Study: Optimizing bits_to_bytes.....	139
Hack 94: Designated Structure Initializers.....	144
Hack 95: Checking printf style Arguments Lists.....	145
Hack 96: Packing structures.....	146
Hack 97: Creating Functions Who's Return Shouldn't Be Ignored.....	146
Hack 98: Creating Functions Which Never Return.....	147
Hack 99: Using the GCC Heap Memory Checking Functions to Locate Errors .....	149
Hack 100: Tracing Memory Usage.....	150
Hack 101: Generating a Backtrace.....	152
Anti-Hack 102: Using "#define extern" for Variable Declarations.....	156
Anti-Hack 103: Use , (comma) to join statements.....	158
Anti-Hack 104: if (strcmp(a,b)) .....	159
Anti-Hack 105: if (ptr) .....	161
Anti-Hack 106: The "while ((ch = getch()) != EOF)" Hack.....	161
Anti-Hack 107: Using #define to Augment the C++ Syntax.....	163
Anti-Hack 108: Using BEGIN and END Instead of { and }.....	163
Anti-Hack 109: Variable Argument Lists.....	164
Anti-Hack 110: Opaque Handles.....	166
Anti-Hack 111: Microsoft (Hungarian) Notation.....	166
Hack 112: Always Verify the Hardware Specification.....	170
Hack 113: Use Portable Types Which Specify Exactly How Wide Your Integers Are.....	171
Hack 114: Verify Structure Sizes.....	172

[Click here to download full PDF material](#)