

Defect  
↑  
Detect



# Memory Thinking

## C & C++

### Linux Diagnostics

Version 2.0

Dmitry Vostokov  
Software Diagnostics Services

# Memory Thinking for C & C++ Linux Diagnostics

---

Slides with Descriptions and Source Code Illustrations

Second Edition

**Dmitry Vostokov**  
**Software Diagnostics Services**

OpenTask

Memory Thinking for C & C++ Linux Diagnostics: Slides with Descriptions and Source Code Illustrations, Second Edition

Published by OpenTask, Republic of Ireland

Copyright © 2024 by OpenTask

Copyright © 2024 by Dmitry Vostokov

Copyright © 2024 by Software Diagnostics Services

Copyright © 2024 by Dublin School of Security

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, transmitted in any form or by any means, or used for training artificial intelligence systems without the prior written permission of the publisher.

OpenTask books are available through booksellers and distributors worldwide. For further information or comments, send requests to [press@opentask.com](mailto:press@opentask.com).

Product and company names mentioned in this book may be trademarks of their owners.

A CIP catalog record for this book is available from the British Library.

ISBN-13: 978-1912636563 (Paperback)

Revision 2.00 (October 2024)

## Table of Contents

<b>Preface</b>	<b>11</b>
<b>About the Author</b>	<b>12</b>
<b>Introduction</b>	<b>13</b>
<i>Original Training Course Name</i>	13
<i>Prerequisites</i>	13
<i>Training Goals</i>	14
<i>Training Principles</i>	14
<i>Schedule</i>	15
<i>Training Idea</i>	15
<i>Version 2.0 Idea</i>	16
<i>General C &amp; C++ Aspects</i>	16
<i>What We Do Not Cover</i>	18
<i>Linux C &amp; C++ Aspects</i>	18
<i>Why C &amp; C++?</i>	19
<i>Which C &amp; C++?</i>	20
<i>My History of C &amp; C++</i>	20
<i>C and C++ Mastery Process</i>	21
<i>Thought Process</i>	22
<b>Philosophy of Pointers</b>	<b>23</b>

<i>Pointer</i>	23
<i>Pointer Dereference</i>	24
<i>One to Many</i>	24
<i>Many to One</i>	25
<i>Many to One Dereference</i>	25
<i>Invalid Pointer</i>	26
<i>Invalid Pointer Dereference</i>	26
<i>Wild (Dangling) Pointer</i>	27
<i>Pointer to Pointer</i>	27
<i>Pointer to Pointer Dereference</i>	28
<i>Naming Pointers and Entities</i>	28
<i>Names as Pointer Content</i>	29
<i>Pointers as Entities</i>	29
<b>Pointer Code Examples</b>	<b>30</b>
<i>Warning</i>	31
<i>Pointer</i>	31
<i>* Placement Style</i>	33
<i>Pointer Dereference</i>	33
<i>One to Many</i>	35
<i>Memory Leak</i>	36

<i>Many to One</i>	38	<i>ASCII Characters and Pointers</i>	61
<i>Many to One Dereference</i>	40	<i>Bytes and Pointers</i>	62
<i>Invalid Pointer</i>	41	<i>Wide Characters and Pointers</i>	63
<i>Invalid Pointer Dereference</i>	43	<i>Integers</i>	64
<i>Wild (Dangling) Pointer</i>	44	<i>Little-Endian System</i>	65
<i>Pointer to Pointer</i>	46	<i>Short Integers</i>	66
<i>Pointer to Pointer Dereference</i>	48	<i>Long and Long Long Integers</i>	67
<i>Undefined Behavior</i>	49	<i>Signed and Unsigned Integers</i>	68
<i>Appendix</i>	50	<i>Fixed Size Integers</i>	69
<b>Memory and Pointers</b>	<b>51</b>	<i>Booleans</i>	70
<i>Mental Exercise</i>	52	<i>Bytes</i>	71
<i>Debugger Memory Layout</i>	52	<i>Alignment (C11)</i>	72
<i>Memory Dereference Layout</i>	53	<i>Alignment (C++11)</i>	73
<i>Names as Addresses</i>	53	<i>Size</i>	74
<i>Addresses and Entities</i>	54	<i>LP64</i>	75
<i>Addresses and Structures</i>	54	<i>Nothing and Anything</i>	76
<i>Pointers to Structures</i>	55	<i>Automatic Type Inference</i>	77
<i>Arrays</i>	55	<i>Appendix</i>	78
<i>Arrays and Pointers to Arrays</i>	56	<b>Entity Conversion</b>	<b>80</b>
<i>Strings and Pointers to Strings</i>	57	<i>Pointer Conversion (C-Style)</i>	81
<i>Appendix</i>	59	<i>Pointer Conversion (C++)</i>	82
<b>Basic Types</b>	<b>60</b>	<i>Numeric Promotion/Conversion</i>	83

<i>Numeric Conversion</i>	84	<i>Appendix</i>	114
<i>Incompatible Types</i>	85	<b>Memory and Structures</b>	<b>116</b>
<i>Forcing</i>	86	<i>Addresses and Structures</i>	117
<i>Uniting</i>	87	<i>Structure Field Access</i>	118
<i>Appendix</i>	89	<i>Pointers to Structures</i>	120
<b>Structures, Classes, and Objects</b>	<b>90</b>	<i>Pointers to Structure Fields</i>	121
<i>Structures</i>	91	<i>Structure Inheritance</i>	124
<i>Access Level</i>	92	<i>Structure Slicing</i>	125
<i>Reading/Writing Private Fields</i>	93	<i>Inheritance Access Level</i>	127
<i>Classes and Objects</i>	94	<i>Structures and Classes II</i>	128
<i>Structures and Classes</i>	95	<i>Reading/Writing Private Base</i>	129
<i>Pointer to Structure</i>	96	<i>Internal Structure Alignment</i>	130
<i>Pointer to Structure Dereference</i>	97	<i>Static Structure Fields</i>	131
<i>One to Many</i>	98	<i>Appendix</i>	132
<i>Memory Leak</i>	100	<b>Uniform Initialization</b>	<b>134</b>
<i>Many Pointers to One Structure</i>	102	<i>Old Initialization Ways</i>	135
<i>Many to One Dereference</i>	103	<i>New Way {}</i>	136
<i>Invalid Pointer to Structure</i>	105	<i>Uniform Structure Initialization</i>	137
<i>Invalid Pointer Dereference</i>	107	<i>Static Field Initialization</i>	139
<i>Wild (Dangling) Pointer</i>	108	<b>Macros, Types, and Synonyms</b>	<b>140</b>
<i>Pointer to Pointer to Structure</i>	111	<i>Macros</i>	141
<i>Pointer to Pointer Dereference</i>	112	<i>Old Way</i>	142

<i>New Way</i>	143	<i>In-place Allocation</i>	163
<b>Enumerations</b>	<b>144</b>	<i>Useful GDB Commands</i>	164
<i>Old Way</i>	145	<i>Appendix</i>	165
<i>New Way</i>	146	<b>Source Code Organisation</b>	<b>166</b>
<i>Appendix</i>	147	<i>Logical Layer (Translation Units)</i>	167
<b>Memory Storage</b>	<b>148</b>	<i>Physical Layer (Source Files)</i>	167
<i>Overview</i>	149	<i>Inter-TU Sharing</i>	168
<i>Memory Regions</i>	149	<i>Classic Static TU Isolation</i>	168
<i>Dynamic Virtual Memory</i>	150	<i>Namespace TU Isolation</i>	169
<i>Static Memory</i>	150	<i>Declaration and Definition</i>	169
<i>Stack Memory</i>	151	<i>TU Definition Conflicts</i>	170
<i>Thread Stack Frames</i>	151	<i>Fine-grained TU Scope Isolation</i>	171
<i>Local Variable Value Lifecycle</i>	152	<i>Conceptual Layer (Design)</i>	172
<i>Scope</i>	154	<i>Incomplete Types</i>	172
<i>Stack Allocation Pitfalls</i>	155	<b>References</b>	<b>175</b>
<i>Explicit Local Allocation</i>	156	<i>Type&amp; vs. Type*</i>	176
<i>Heap Memory</i>	157	<b>Values</b>	<b>177</b>
<i>Dynamic Allocation (C-style)</i>	158	<i>Value Categories</i>	178
<i>Dynamic Allocation (C++)</i>	158	<i>Constant Values</i>	179
<i>Memory Expressions</i>	159	<i>Constant Expressions</i>	181
<i>Memory Operators</i>	159	<b>Functions</b>	<b>182</b>
<i>Local Pointers (Manual)</i>	162	<i>Macro Functions</i>	183

<i>constexpr Functions</i>	184	<i>Structure Destructor Hierarchy</i>	213
<i>Pointers to Functions</i>	185	<i>Structure Virtual Destructor</i>	214
<i>Function Pointer Types</i>	186	<i>Structure Member Destruction</i>	216
<i>Reading Declarations</i>	187	<i>Destructor as a Method</i>	217
<i>Structure Function Fields</i>	188	<i>Structure Default Operations</i>	218
<i>Structure Methods</i>	189	<i>Structure Deleted Operations</i>	219
<i>Structure Methods (Inlined)</i>	190	<i>Conversion Operators</i>	221
<i>Structure Methods (Inheritance)</i>	191	<i>Parameters by Value</i>	222
<i>Structure Virtual Methods</i>	192	<i>Parameters by Pointer/Reference</i>	224
<i>Structure Pure Virtual Methods</i>	195	<i>Parameters by Ptr/Ref to Const</i>	226
<i>Structure as Interface</i>	197	<i>Parameters by Ref to Rvalue</i>	228
<i>Function Structure</i>	199	<i>Possible Mistake</i>	229
<i>Structure Constructors</i>	200	<i>Function Overloading</i>	229
<i>Structure Converting Constructors</i>	201	<i>Default Arguments</i>	230
<i>Structure Delegating Constructors</i>	202	<i>Variadic Functions</i>	231
<i>Structure Member Initialization</i>	204	<i>Immutable Objects</i>	232
<i>Structure Copy Constructor</i>	205	<i>Static Structure Functions</i>	233
<i>Copy vs. Move Semantics</i>	206	<i>Lambdas</i>	234
<i>Structure Move Constructors</i>	207	<i>x64 CPU Registers</i>	234
<i>Structure Copy Assignment</i>	208	<i>x64 Instructions and Registers</i>	235
<i>Structure Move Assignment</i>	210	<i>x64 Memory and Stack Addressing</i>	235
<i>Structure Destructor</i>	212	<i>x64 Memory Load Instructions</i>	236



<i>x64 Memory Store Instructions</i>	236	<i>std::function Lambda Parameters</i>	251
<i>x64 Flow Instructions</i>	237	<i>auto Lambda Parameters</i>	252
<i>x64 Function Parameters</i>	237	<i>Lambdas as Return Values</i>	253
<i>x64 Struct Function Parameters</i>	238	<i>Appendix</i>	254
<i>A64 CPU Registers</i>	238	<b>Virtual Function Call</b>	<b>257</b>
<i>A64 Instructions and Registers</i>	239	<i>VTBL Memory Layout</i>	258
<i>A64 Memory and Stack Addressing</i>	239	<i>VPTR and Struct Memory Layout</i>	258
<i>A64 Memory Load Instructions</i>	240	<b>Templates: A Planck-length Introduction</b>	<b>260</b>
<i>A64 Memory Store Instructions</i>	240	<i>Why Templates?</i>	261
<i>A64 Flow Instructions</i>	241	<i>Reusability</i>	261
<i>A64 Function Parameters</i>	241	<i>Types of Templates</i>	262
<i>A64 Struct Function Parameters</i>	242	<i>Types of Template Parameters</i>	263
<i>this</i>	243	<i>Type Safety</i>	264
<i>Function Objects vs. Lambdas</i>	244	<i>Flexibility</i>	265
<i>A64 Lambda Example</i>	245	<i>Metafunctions</i>	266
<i>Captures and Closures</i>	245	<i>Variadic Templates</i>	267
<i>A64 Captures Examples</i>	246	<b>Iterators as Pointers</b>	<b>268</b>
<i>Lambdas as Parameters</i>	247	<i>Containers</i>	269
<i>A64 Lambda Parameter Example</i>	248	<i>Iterators</i>	269
<i>Lambda Parameter Optimization</i>	248	<i>Constant Iterators</i>	270
<i>A64 Optimization Example</i>	249	<i>Pointers as Iterators</i>	271
<i>Lambdas as Unnamed Functions</i>	250	<i>Algorithms</i>	272

<b>Memory Ownership</b>	<b>274</b>	<i>File Descriptor RAII</i>	283
<i>Pointers as Owners</i>	275	<b>Threads and Synchronization</b>	<b>285</b>
<i>Problems with Pointer Owners</i>	275	<i>Threads in C/C++</i>	286
<b>Smart Pointers</b>	<b>276</b>	<i>Threads in C++ Proper</i>	287
<i>Basic Design</i>	277	<i>Synchronization Problems</i>	288
<i>Unique Pointers</i>	277	<i>Synchronization Solution</i>	289
<i>Descriptors as Unique Pointers</i>	278	<b>Memory-safe C++ Development</b>	<b>290</b>
<i>Shared Pointers</i>	279	<i>URSS Principle</i>	290
<b>RAII</b>	<b>281</b>	<b>Resources</b>	<b>291</b>
<i>RAII Definition</i>	282	<i>C and C++</i>	291
<i>RAII Advantages</i>	282	<i>Training (Linux C and C++)</i>	292