Memory Thinking for C & C++
Windows Diagnostics

Slides with Descriptions Only

Dmitry Vostokov
Software Diagnostics Services

OpenTask
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>Preface</td>
<td>15</td>
</tr>
<tr>
<td>About the Author</td>
<td>16</td>
</tr>
<tr>
<td>Introduction</td>
<td>17</td>
</tr>
<tr>
<td><em>Original Training Course Name</em></td>
<td>17</td>
</tr>
<tr>
<td><em>Prerequisites</em></td>
<td>18</td>
</tr>
<tr>
<td><em>Training Goals</em></td>
<td>19</td>
</tr>
<tr>
<td><em>Training Principles</em></td>
<td>20</td>
</tr>
<tr>
<td><em>Schedule</em></td>
<td>21</td>
</tr>
<tr>
<td><em>Training Idea</em></td>
<td>22</td>
</tr>
<tr>
<td><em>General C &amp; C++ Aspects</em></td>
<td>23</td>
</tr>
<tr>
<td><em>What We Do Not Cover</em></td>
<td>25</td>
</tr>
<tr>
<td><em>Windows C &amp; C++ Aspects</em></td>
<td>26</td>
</tr>
<tr>
<td>*Why C &amp; C++?</td>
<td>27</td>
</tr>
<tr>
<td>*Which C &amp; C++?</td>
<td>29</td>
</tr>
<tr>
<td><em>My History of C &amp; C++</em></td>
<td>30</td>
</tr>
<tr>
<td><em>C and C++ Mastery Process</em></td>
<td>32</td>
</tr>
<tr>
<td><em>Thought Process</em></td>
<td>33</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Philosophy of Pointers</td>
<td>34</td>
</tr>
<tr>
<td>Pointer</td>
<td>35</td>
</tr>
<tr>
<td>Pointer Dereference</td>
<td>36</td>
</tr>
<tr>
<td>Many to One</td>
<td>37</td>
</tr>
<tr>
<td>Many to One Dereference</td>
<td>38</td>
</tr>
<tr>
<td>Invalid Pointer</td>
<td>39</td>
</tr>
<tr>
<td>Invalid Pointer Dereference</td>
<td>40</td>
</tr>
<tr>
<td>Wild (Dangling) Pointer</td>
<td>41</td>
</tr>
<tr>
<td>Pointer to Pointer</td>
<td>42</td>
</tr>
<tr>
<td>Pointer to Pointer Dereference</td>
<td>43</td>
</tr>
<tr>
<td>Naming Pointers and Entities</td>
<td>44</td>
</tr>
<tr>
<td>Names as Pointer Content</td>
<td>45</td>
</tr>
<tr>
<td>Pointers as Entities</td>
<td>46</td>
</tr>
<tr>
<td>Memory and Pointers</td>
<td>47</td>
</tr>
<tr>
<td>Mental Exercise</td>
<td>48</td>
</tr>
<tr>
<td>Debugger Memory Layout</td>
<td>49</td>
</tr>
<tr>
<td>Memory Dereference Layout</td>
<td>50</td>
</tr>
<tr>
<td>Names as Addresses</td>
<td>51</td>
</tr>
<tr>
<td>Addresses and Entities</td>
<td>52</td>
</tr>
</tbody>
</table>
Addresses and Structures

Pointers to Structures

Arrays

Arrays and Pointers to Arrays

Strings and Pointers to Strings

Basic Types

ASCII Characters and Pointers

Bytes and Pointers

Wide Characters and Pointers

Integers

Long Integers

Little-Endian System

Short Integers

Long Long Integers

Signed and Unsigned Integers

Fixed Size Integers

Booleans

Bytes

Size
Alignment 72

LLP64 73

Nothing and Anything 74

Automatic Type Inference 75

**Entity Conversion** 76

Pointer Conversion (C-Style) 77

Numeric Promotion/Conversion 78

Numeric Conversion 79

Incompatible Types 80

Forcing 81

**Structures, Classes, and Objects** 83

Structures 84

Access Level 85

Classes and Objects 86

Structures and Classes 87

Pointer to Structure 88

Pointer to Structure Dereference 89

Many Pointers to One Structure 90

Many to One Dereference 91
Invalid Pointer to Structure  
Invalid Pointer Dereference  
Wild (Dangling) Pointer  
Pointer to Pointer to Structure  
Pointer to Pointer Dereference  

Memory and Structures  
Addresses and Structures  
Structure Field Access  
Pointers to Structures  
Pointers to Structure Fields  
Structure Inheritance  
Structure Slicing  
Inheritance Access Level  
Structures and Classes II  
Internal Structure Alignment  
Static Structure Fields  

Uniform Initialization  
Old Initialization Ways  
New Way {}
Uniform Structure Initialization 112

Static Field Initialization 113

Macros, Types, and Synonyms 114

Macros 115

Old Way 116

New Way 117

Memory Storage 118

Overview 119

Thread Stack Frames 120

Local Variable Value Lifecycle 121

Stack Allocation Pitfalls 123

Explicit Local Allocation 125

Dynamic Allocation (C-style) 126

Dynamic Allocation (C++) 127

Memory Operators 128

Memory Expressions 129

Local Pointers (Manual) 130

In-place Allocation 132

Source Code Organisation 133
Logical Layer (Translation Units) 134

Physical Layer (Source Files) 135

Inter-TU Sharing 136

Classic Static TU Isolation 137

Namespace TU Isolation 138

Declaration and Definition 139

TU Definition Conflicts 140

Fine-grained TU Scope Isolation 141

Conceptual Layer (Design) 142

Incomplete Types 143

References 144

Type& vs. Type* 145

Values 146

Value Categories 147

Constant Values 148

Constant Expressions 149

Functions 150

Pointers to Functions 151

Function Pointer Types 153
Reading Declarations 154
Structure Function Fields 155
Structure Methods 156
Structure Methods (Inlined) 157
Structure Methods (Inheritance) 158
Structure Virtual Methods 160
Structure Pure Virtual Methods 162
Structure as Interface 163
Function Structure 164
Structure Constructors 165
Structure Copy Constructor 166
Structure Copy Assignment 167
Structure Destructor 168
Structure Destructor Hierarchy 169
Structure Virtual Destructor 170
Destructor as a Method 171
Conversion Operators 172
Parameters by Value 174
Parameters by Pointer/Reference 175
Parameters by Ptr/Ref to Const
Possible Mistake
Function Overloading
Immutable Objects
Static Structure Functions
Lambdas
x64 CPU Registers
Instructions and Registers
Memory and Stack Addressing
Memory Cell Sizes
Memory Load Instructions
Memory Store Instructions
Flow Instructions
Function Parameters
Struct Function Parameters
this
Function Objects vs. Lambdas
Captures and Closures
Lambdas as Parameters
Lambda Parameter Optimization

Lambdas as Unnamed Functions

std::function Lambda Parameters

auto Lambda Parameters

Lambdas as Return Values

Virtual Function Call

VTBL Memory Layout

VPTR and Struct Memory Layout

Templates: A Planck-length Introduction

Why Templates?

Reusability

Types of Templates

Types of Template Parameters

Type Safety

Flexibility

Metafunctions

Iterators as Pointers

Containers

Iterators
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Iterators</td>
<td>223</td>
</tr>
<tr>
<td>Pointers as Iterators</td>
<td>224</td>
</tr>
<tr>
<td>Algorithms</td>
<td>225</td>
</tr>
<tr>
<td><strong>Memory Ownership</strong></td>
<td>226</td>
</tr>
<tr>
<td>Pointers as Owners</td>
<td>227</td>
</tr>
<tr>
<td>Problems with Pointer Owners</td>
<td>228</td>
</tr>
<tr>
<td><strong>Smart Pointers</strong></td>
<td>229</td>
</tr>
<tr>
<td>Basic Design</td>
<td>230</td>
</tr>
<tr>
<td>Unique Pointers</td>
<td>231</td>
</tr>
<tr>
<td>Handles as Unique Pointers</td>
<td>232</td>
</tr>
<tr>
<td>Shared Pointers</td>
<td>233</td>
</tr>
<tr>
<td><strong>RAII</strong></td>
<td>234</td>
</tr>
<tr>
<td>RAII Definition</td>
<td>235</td>
</tr>
<tr>
<td>RAII Advantages</td>
<td>236</td>
</tr>
<tr>
<td>Handle RAII</td>
<td>237</td>
</tr>
<tr>
<td><strong>Threads and Synchronization</strong></td>
<td>238</td>
</tr>
<tr>
<td>Threads in C/C++</td>
<td>239</td>
</tr>
<tr>
<td>Threads in C++ Proper</td>
<td>240</td>
</tr>
<tr>
<td>Synchronization Problems</td>
<td>241</td>
</tr>
</tbody>
</table>
Synchronization Solution 242

Resources 243

C and C++ 244

Reading Windows-based Code 245

Windows (C and C++) 247

Training (Windows C and C++) 249