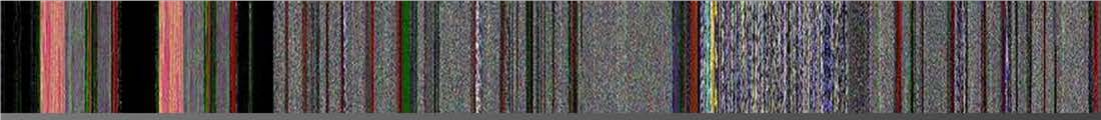

Memory Thinking

C & C++

Linux Diagnostics

Dmitry Vostokov
Software Diagnostics Services

Memory Thinking for C & C++ Linux Diagnostics

Slides with Descriptions Only

Dmitry Vostokov
Software Diagnostics Services

OpenTask

Memory Thinking for C & C++ Linux Diagnostics: Slides with Descriptions Only

Published by OpenTask, Republic of Ireland

Copyright © 2023 by OpenTask

Copyright © 2023 by Dmitry Vostokov

Copyright © 2023 by Software Diagnostics Services

Copyright © 2023 by Dublin School of Security

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means 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.

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: 978-1912636570 (Paperback)

Revision 1.00 (December 2023)

Table of Contents

Table of Contents **3**

Preface **15**

About the Author **16**

Introduction **17**

Original Training Course Name *17*

Prerequisites *18*

Training Goals *19*

Training Principles *20*

Schedule *21*

Training Idea *22*

General C & C++ Aspects *23*

What We Do Not Cover *25*

Linux C & C++ Aspects *26*

Why C & C++? *27*

Which C & C++? *29*

My History of C & C++ *30*

C and C++ Mastery Process *32*

Thought Process *33*

Philosophy of Pointers **34**

Pointer *35*

Pointer Dereference *36*

Many to One *37*

Many to One Dereference *38*

Invalid Pointer *39*

Invalid Pointer Dereference *40*

Wild (Dangling) Pointer *41*

Pointer to Pointer *42*

Pointer to Pointer Dereference *43*

Naming Pointers and Entities *44*

Names as Pointer Content *45*

Pointers as Entities *46*

Memory and Pointers **47**

Mental Exercise *48*

Debugger Memory Layout *49*

Memory Dereference Layout *50*

Names as Addresses *51*

Addresses and Entities *52*

Addresses and Structures *53*

Pointers to Structures *54*

Arrays *55*

Arrays and Pointers to Arrays *56*

Strings and Pointers to Strings *57*

Basic Types **58**

ASCII Characters and Pointers *59*

Bytes and Pointers *60*

Wide Characters and Pointers *61*

Integers *62*

Little-Endian System *63*

Short Integers *64*

Long and Long Long Integers *65*

Signed and Unsigned Integers *66*

Fixed Size Integers *67*

Booleans *68*

Bytes *69*

Size *70*

Alignment *71*

LP64 *72*

Nothing and Anything *73*

Automatic Type Inference *74*

Entity Conversion **75**

Pointer Conversion (C-Style) *76*

Numeric Promotion/Conversion *77*

Numeric Conversion *78*

Incompatible Types *79*

Forcing *80*

Structures, Classes, and Objects **82**

Structures *83*

Access Level *84*

Classes and Objects *85*

Structures and Classes *86*

Pointer to Structure *87*

Pointer to Structure Dereference *88*

Many Pointers to One Structure *89*

Many to One Dereference *90*

Invalid Pointer to Structure *91*

Invalid Pointer Dereference *92*

Wild (Dangling) Pointer *93*

Pointer to Pointer to Structure *94*

Pointer to Pointer Dereference *95*

Memory and Structures **96**

Addresses and Structures *97*

Structure Field Access *98*

Pointers to Structures *99*

Pointers to Structure Fields *100*

Structure Inheritance *101*

Structure Slicing *102*

Inheritance Access Level *104*

Structures and Classes II *105*

Internal Structure Alignment *106*

Static Structure Fields *107*

Uniform Initialization **108**

Old Initialization Ways *109*

New Way {} *110*

Uniform Structure Initialization *111*

Static Field Initialization *112*

Macros, Types, and Synonyms **113**

Macros *114*

Old Way *115*

New Way *116*

Memory Storage **117**

Overview *118*

Thread Stack Frames *119*

Local Variable Value Lifecycle *120*

Stack Allocation Pitfalls *122*

Explicit Local Allocation *124*

Dynamic Allocation (C-style) *125*

Dynamic Allocation (C++) *126*

Memory Operators *127*

Memory Expressions *128*

Local Pointers (Manual) *129*

In-place Allocation *131*

Source Code Organisation **132**

Logical Layer (Translation Units) *133*

Physical Layer (Source Files) *134*

Inter-TU Sharing *135*

Classic Static TU Isolation *136*

Namespace TU Isolation *137*

Declaration and Definition *138*

TU Definition Conflicts *139*

Fine-grained TU Scope Isolation *140*

Conceptual Layer (Design) *141*

Incomplete Types *142*

References **143**

*Type& vs. Type** *144*

Values **145**

Value Categories *146*

Constant Values *147*

Constant Expressions *148*

Functions **149**

Pointers to Functions *150*

Function Pointer Types *152*

Reading Declarations *153*

Structure Function Fields *154*

Structure Methods *155*

Structure Methods (Inlined) *156*

Structure Methods (Inheritance) *157*

Structure Virtual Methods *159*

Structure Pure Virtual Methods *161*

Structure as Interface *162*

Function Structure *163*

Structure Constructors *164*

Structure Copy Constructor *165*

Structure Copy Assignment *166*

Structure Destructor *167*

Structure Destructor Hierarchy *168*

Structure Virtual Destructor *169*

Destructor as a Method *170*

Conversion Operators *171*

Parameters by Value *173*

Parameters by Pointer/Reference *174*

Parameters by Ptr/Ref to Const *175*

Possible Mistake *176*

Function Overloading *177*

Immutable Objects *178*

Static Structure Functions *179*

Lambdas *180*

x64 CPU Registers *181*

x64 Instructions and Registers *182*

x64 Memory and Stack Addressing *183*

x64 Memory Load Instructions *184*

x64 Memory Store Instructions *185*

x64 Flow Instructions *186*

x64 Function Parameters *187*

x64 Struct Function Parameters *188*

A64 CPU Registers *189*

A64 Instructions and Registers *190*

A64 Memory and Stack Addressing *191*

A64 Memory Load Instructions *192*

A64 Memory Store Instructions *193*

A64 Flow Instructions *194*

A64 Function Parameters *195*

A64 Struct Function Parameters *196*

this *197*

Function Objects vs. Lambdas *198*

A64 Lambda Example *200*

Captures and Closures *201*

A64 Captures Example *203*

Lambdas as Parameters *204*

A64 Lambda Parameter Example *206*

Lambda Parameter Optimization *207*

A64 Optimization Example *209*

Lambdas as Unnamed Functions *210*

std::function Lambda Parameters *212*

auto Lambda Parameters *214*

Lambdas as Return Values *216*

Virtual Function Call **218**

VTBL Memory Layout *219*

VPTR and Struct Memory Layout *220*

Templates: A Planck-length Introduction **221**

Why Templates? *222*

Reusability *223*

Types of Templates *225*

Types of Template Parameters *226*

Type Safety *228*

Flexibility *230*

Metafunctions *231*

Iterators as Pointers **232**

Containers *233*

Iterators *234*

Constant Iterators *235*

Pointers as Iterators *236*

Algorithms *237*

Memory Ownership **238**

Pointers as Owners *239*

Problems with Pointer Owners *240*

Smart Pointers **241**

Basic Design *242*

Unique Pointers *243*

Descriptors as Unique Pointers *244*

Shared Pointers *245*

RAII **246**

RAII Definition *247*

RAII Advantages *248*

File Descriptor RAII *249*

Threads and Synchronization **250**

Threads in C/C++ *251*

Threads in C++ Proper *252*

Synchronization Problems *253*

Synchronization Solution *254*

Resources **255**

C and C++ *256*

Training (Linux C and C++) *257*