C & C++
Windows Diagnostics
Accelerated

Version 2.0

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

- Development experience

and (optional)

- Basic memory dump analysis
Training Goals

- Review common fundamentals of C and C++
- Review C++ specifics
- Use WinDbg for learning C and C++ internals
- See how C and C++ knowledge is used during diagnostics and debugging
Training Principles

- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content and examples
std::vector<Session> sessions;
assert(sessions.size() == 6);
assert(sessions.capacity() > 6);
Training Idea

- Reading Windows-based Code training
- Memory dump analysis training
- Reversing training
- Windows API training
Version 2.0 Idea

- Missing topics
- Missing source code projects
- Memory Thinking for Rust training
General C & C++ Aspects

- Philosophy of pointers
- Structures, classes, and objects
- Promotions and conversions
- Macros, types, and synonyms
- Source code organization, PImpl
- Pointer dereference walkthrough
- Functions and function pointers
- Inheritance
- Operators, function objects
- Destructors, virtual destructors
- Local stack variables and values
- Memory operators and expressions
- Alignment
- Slicing
- Iterators as pointers
- Lambdas and their internals
- Threads and synchronization
- Deleted and default members
- Unions
- Variadic functions and templates

- Memory and pointers
- Basic types
- Memory and structures
- Uniform initialization
- Memory storage
- Values, Ivalues, rvalues
- References (Ivalue and rvalue)
- Constant values and expressions
- Namespaces
- Constructors, copy, move, assignment
- Virtual functions, pure methods
- Vtbl and VPTR
- Access levels
- Overloading, overriding
- Templates
- Memory ownership, RAII
- Smart pointers
- Conversion constructors
- Enumerations

© 2024 Software Diagnostics Services
What We Do Not Cover*

- Legacy C
- Concepts
- Coroutines
- Modules
- Tasks
- Ranges
- Container and algorithm semantics and pragmatics
- Container allocators
- Polymorphic allocators

* We promise to include some of these topics in the third edition
Windows C & C++ Aspects

- Windows-specific type aliases and macros
- Desktop application walkthrough
- Desktop application improvement
- Service walkthrough
- Command-line application walkthrough
- LLP64
- COM
- Necessary x64 disassembly
- Parameter passing
- Implicit parameter
Why C & C++?

- Interfacing
- Malware analysis
- Vulnerability analysis and exploitation
- Reversing
- Diagnostics
- Low-level debugging
- OS Monitoring
- Memory forensics
- Crash and hang analysis
- Secure coding
- Static code analysis
- Trace and log analysis
Which C & C++?

- C (C11)
- C++ as a better C
- Classic C++ (C++98, C++03)
- Modern C++ (C++11 ... C++23)
- Windows specifics

Proper C++
My History of C & C++

- C from 1987 and C++ from 1989 (Old CV)
- C++ as a better C from 1991
- Implicit design patterns in 1994-1995
- C++ as proper C++ from 2000
- Explicit design patterns in 2000
- C++98/03/STL from 2001
- Windows memory dump analysis from 2003
- [...]  
- C++11/14 from 2016
- C++17 from 2017
- Functional programming from 2020
- C++20 from 2023
- C++23 from 2024
C and C++ Mastery Process

Coding

Mental Compiling
## Thought Process

- **C and C++** | **Memory**
- **Scala/FP**  | **Functions**
- **Python**    | **Data**
Philosophy of Pointers
Pointer
Pointer Dereference
One to Many

Pointer

Pointer

© 2024 Software Diagnostics Services
Many to One
Many to One Dereference

Diagram: Pointer - Box - Pointer
Invalid Pointer
Invalid Pointer Dereference
Wild (Dangling) Pointer
Pointer to Pointer
Pointer to Pointer Dereference
Naming Pointers and Entities

A
1
fadb6810

B
2
86556810

C
3
a656ffbd
Names as Pointer Content

fadb6810 86556810 a656ffbd

86556810 a656ffbd
Pointers as Entities

fadb6810

86556810

86556810

a656ffbd

a656ffbd

00000000
Pointer Code Examples
Warning

Because of live execution, due to differences in actual systems and ASLR (Address Space Layout Randomization), when you run applications, actual addresses in the output may differ from those shown in the slide description output.
Pointer

```c
int* p = &n;
```
* Placement Style

- `int *p; // first edition`
- `int* p; // second edition`
- `int * p;`
Pointer Dereference

```c
assert(*p == n);
```
One to Many

```c
int* p = &n;
p = &m;
```
Memory Leak

```
int* p = (int*)malloc(...);
```

```
int* p = (int*)malloc(...);
```

```
p = (int*)malloc(...);
```

```
p = (int*)malloc(...);
```
Many to One

```c
int* p1 = &n;
int* p2 = &n;
```
Many to One Dereference

assert(*p1 == n);  assert(*p2 == n);

assert(*p1 == *p2);
Invalid Pointer

```
int* p = (int*)0xffffffff00000000;
```
Invalid Pointer Dereference

```c
int* p = (int*)4;
*p = 0;
```
Wild (Dangling) Pointer

```c
int* p = (int*)malloc(...);
free(p);

* p = 0;
```
Pointer to Pointer

```c
int* p = &n;
int** pp = &p;
```
assert(*pp == p);
assert(**pp == n);

assert(*pp == &n);
Undefined Behavior

- OK | Corruption | Crash | Spike | Hang | Leak
- Different on different machines
- Different at different times
- Depends on compiler-generated code
- Depends on memory layout
Memory and Pointers
Mental Exercise

How many pointers can you count?

<table>
<thead>
<tr>
<th>2ab1000</th>
<th>2ab1004</th>
<th>2ab1008</th>
<th>2ab100c</th>
<th>2ab1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ab1008</td>
<td>ffffffff</td>
<td>2ab1010</td>
<td>2ab100c</td>
<td>00000000</td>
</tr>
</tbody>
</table>
Memory Dereference Layout

```
00002000
00000000
2ab1008
2ab1000: 2ab1008
2ab1004: ffffffff
2ab1008: 2ab1010
2ab100c: 2ab100c
2ab1010: 00000000
2ab1014: 00002000
```

```
2ab1000: 2ab1008 2ab1010
2ab1004: ffffffff ????????
2ab1008: 2ab1010 00000000
2ab100c: 2ab100c 2ab100c
2ab1010: 00000000 ????????
2ab1014: 00002000 ????????
```

© 2024 Software Diagnostics Services
Names as Addresses

2ab1000: 2ab1008
2ab1004: ffffffff
2ab1008: 2ab1010
2ab100c: 2ab100c
2ab1010: 00000000
2ab1014: 00002000
# Addresses and Entities

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ab1000</td>
<td>2ab1008</td>
</tr>
<tr>
<td>2ab1004</td>
<td>ffffffff</td>
</tr>
<tr>
<td>2ab1008</td>
<td>2ab1010</td>
</tr>
<tr>
<td>2ab100c</td>
<td>2ab100c</td>
</tr>
<tr>
<td>2ab1010</td>
<td>00000000</td>
</tr>
<tr>
<td>2ab1014</td>
<td>00002000</td>
</tr>
</tbody>
</table>

© 2024 Software Diagnostics Services
Addresses and Structures

2ab1000: 2ab1008
2ab1004: ffffffff
2ab1008: 2ab1010
2ab1010: 2ab100c
2ab1010: 00000000
2ab1014: 00002000
Pointers to Structures

2ab1000: 2ab1008
2ab1004: fffffffff
2ab1008: 2ab1010
2ab1010: 2ab100c
2ab1014: 00000000

2ab1216: 2ab1004
# Arrays

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ab1000:</td>
<td>2ab1008</td>
</tr>
<tr>
<td>2ab1004:</td>
<td>ffffffff</td>
</tr>
<tr>
<td>2ab1008:</td>
<td>2ab1010</td>
</tr>
<tr>
<td>2ab100c:</td>
<td>2ab100c</td>
</tr>
<tr>
<td>2ab1010:</td>
<td>00000000</td>
</tr>
<tr>
<td>2ab1014:</td>
<td>00002000</td>
</tr>
</tbody>
</table>

- The diagram shows the memory layout of the arrays with the values highlighted.
Arrays and Pointers to Arrays
Strings and Pointers to Strings

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ab1000</td>
<td>'H'</td>
</tr>
<tr>
<td>2ab1001</td>
<td>'e'</td>
</tr>
<tr>
<td>2ab1002</td>
<td>'l'</td>
</tr>
<tr>
<td>2ab1003</td>
<td>'l'</td>
</tr>
<tr>
<td>2ab1004</td>
<td>'o'</td>
</tr>
<tr>
<td>2ab1005</td>
<td>00</td>
</tr>
</tbody>
</table>

2ab1216: 2ab1000
Memory and Pointers
Code Examples