//-----------------------------------------------------------------------------

Edit in MemoryLeakIndicatorMainpage.docx

//-----------------------------------------------------------------------------

/\*! \mainpage **MemoryLeakIndicator 0.0.2**

\if english

# \section **MLI**\_INTRODUCTION Introduction

If there are memory leaks again, just link the DLL from the <b>**MemoryLeakIndicator**</b> to the project (I use the <b>MemoryLeakIndicator</b> only in debug mode).

In the main file include <b>**MliForMain.h**</b>, in the implementation files where something is instantiated with new, after all other includes only the <b>**MilDebugNew.h**</b>.

In the respective project the <b>MilNewDelete.cpp</b> must be compiled as well.

The English text was translated with the support of <a target="\_blanc" rel="noopener noreferrer" href="http://www.DeepL.com/Translator">www.DeepL.com</a>.

\endif

\if german

# \section **MLI**\_INTRODUCTION Einleitung

Wenn es mal wieder Memory-Leaks gibt, einfach die DLL vom <b>**MemoryLeakIndicator**</b> zum Projekt linken.

In der Main bindet man <b>**MliForMain.h**</b> ein, in den Implementierungsdateien, in denen was mit new instantiiert wird, hinter allen anderen Includes nur die <b>**MilDebugNew.h**</b>.

Im jeweiligen Projekt muss die <b>**MilNewDelete.cpp**</b> mit kompiliert werden.

\endif

\if english

# \section **MLI\_MOTIVATION Motivation**

I use with Visual Studio on Windows the <b>**crtdbg.h**</b> to get memory leaks displayed.

**\sa** https://github.com/MicrosoftDocs/visualstudio-docs/blob/main/docs/debugger/finding-memory-leaks-using-the-crt-library.md

Unfortunately, the file name and the line number are only displayed in the Visual Studio output-window when using <b>**malloc**</b>, but not when using <b>**new**</b>. In the case of memory leaks caused by a <b>**new**</b>, only the address and the data are displayed. While it is possible to assign the value from the curly brackets to the variable <b>**\_crtBreakAlloc**</b> in the hope that the program stops at the place where the non-deleted object is instantiated. But this is unfortunately not always reliable (e.g. with DLLs), since if necessary always another memory location is taken.

Finding the causing instructions in the source code is sometimes difficult. I would like to have the file name and the line number as a hint.

\endif

\if german

# \section **MLI\_MOTIVATION Motivation**

Ich nutze mit Visual Studio unter Windows die <b>**crtdbg.h**</b> um Memory-Leaks angezeigt zu bekommen.

**\sa** https://docs.microsoft.com/de-de/visualstudio/debugger/finding-memory-leaks-using-the-crt-library?view=vs-2022

Leider werden der Dateiname und die Zeilennummer nur bei Benutzung von <b>**malloc**</b>, aber nicht bei Nutzung von <b>**new**</b> in der Ausgabe angezeigt. Bei Memory-Leaks verursacht durch ein <b>**new**</b>, werden nur die Adresse und die Daten angezeigt. Es ist zwar möglich, den Wertes aus den geschweiften Klammern der Variable <b>**\_crtBreakAlloc**</b> zuzuweisen, in der Hoffnung, dass das Programm bei der Ausführung an der Stelle anhält, wo das nicht gelöschte Objekt instantiiert wird. Aber dieses ist leider nicht immer (z.B. bei DLLs) zuverlässig, da ggf. immer ein anderer Speicherort genommen wird.

Das Finden der verursachenden Anweisungen im Sourcecode gestaltet sich bisweilen schwierig. Gerne hätte hätte ich den Dateinamen und die Zeilennummer als Hinweis.

\endif

\if english

# \section **MLI\_IMPLEMENTATION implementation**

If there are memory leaks again, just link the DLL from the <b>**MemoryLeakIndicator**</b> to the project.

In addition, the source file <b>**MilNewDelete.cpp**</b> (with the overloaded operators new and delete) must also be compiled in the project.

This also applies to self-created DLLs if memory leaks are to be detected there.

In the main file include <b>**MliForMain.h**</b>, in the implementation files where something is instantiated with new, after all other includes only the <b>**MilDebugNew.h**</b>. I enclose this with a preprocessor statement such as:

\endif

\if german

# \section **MLI\_IMPLEMENTATION Implementierung**

Wenn es mal wieder Memory-Leaks gibt, einfach die DLL vom <b>**MemoryLeakIndicator**</b> zum Projekt linken.

Zusätzlich muss im Projekt die Quellatei <b>**MilNewDelete.cpp**</b> (mit den überladenen Operator new und delete) mit kompiliert werden.

Das gilt auch für selbsterstellte DLLs, wenn dort Memory-Leaks erkannt werden sollen.

In der Main bindet man <b>**MliForMain.h**</b> ein, in den Implementierungsdateien, in denen was mit new instantiiert wird, hinter allen anderen Includes nur die <b>**MilDebugNew.h**</b>. Dieses umschließe ich mit einer Präprozessor-Anweisung wie z.B.:

\endif

\code{.cpp}

#if defined \_MSC\_VER && defined \_DEBUG\_MEMORY\_SPY

...

#endif

\endcode

\if english

1. **\note** It is not crowned with success to take over all sources of <b>**MemoryLeakIndicator**</b> into the own project, thus not to use the DLL.
2. \endif
3. \if german
4. **\note** Es ist nicht von Erfolg gekrönt, alle Sourcen von <b>**MemoryLeakIndicator**</b> in das eigene Projekt zu übernehmen, also die DLL nicht zu benutzen.
5. \endif

\if english

# \section **MLI\_MAIN\_EX Example**

Here is a small example:

\endif

\if german

# \section **MLI\_MAIN\_EX Beispiel**

Im folgenden ein kleines Beispiel:

\endif

\code{.cpp}

1 #if defined \_MSC\_VER && defined \_DEBUG

2 // detect Memory-Leaks

3 #define \_CRTDBG\_MAP\_ALLOC

4 #include <crtdbg.h>

5 #endif

6

7 #include <iostream>

8 #include <string>

9 #include <conio.h>

10 #if defined \_MSC\_VER && defined \_DEBUG\_MEMORY\_SPY

11 #include <MliForMain.h>

12 #endif

13 //-----------------------------------------------------------------------------

14 class MyClass

15 {

16 private:

17 int \* m\_piT1;

18 public:

19 MyClass( int i )

20 {

21 m\_piT1 = new int (i);

22 };

23 ~MyClass()

24 {

25 // Oh, m\_piT1 forgot to delete

26 //delete m\_piT1;

27 };

28 };

29 //-----------------------------------------------------------------------------

30 int main(int argc, char \*argv[])

31 {

32 #if defined \_MSC\_VER && defined \_DEBUG

33 // Displays memory leaks when there are any...

34 \_CrtSetDbgFlag ( \_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF );

35 // Comment out next line with number from { } -> e.g..: "{173} normal block at 0x00D01C78, 10 bytes long."

36 //\_crtBreakAlloc = 172;

37 // Is unfortunately not always (e.g. with DLLs) reliably, because if necessary other memory location.

38 #endif

39

40 #if defined \_MSC\_VER && defined \_DEBUG\_MEMORY\_SPY

41 MemSpy::activateMemoryMonitoring();

42 #endif

43

44 // Examples:

45 MyClass \* t\_pClass = new MyClass( 1 );

46 //delete t\_pClass;

47

48 int \* t\_piLeak = new int( 2 );

49 //delete t\_piLeak;

50

51 char \* t\_szLeak = new char[100];

52 //delete[] t\_szLeak;

53

54 #if defined \_MSC\_VER && defined \_DEBUG\_MEMORY\_SPY

55 // Display memory leaks

56 MemSpy::finishMemoryMonitoring( MemSpy::OUTPUT\_TRACE | MemSpy::OUTPUT\_CONSOLE );

57 #endif

58 std::cout << "press any key . . .";

59 \_getch();

60 return 0;

61 }

62 //-----------------------------------------------------------------------------

\endcode

\if english

Since the parameters <b>**MemSpy::OUTPUT\_TRACE**</b> and <b>**MemSpy::OUTPUT\_CONSOLE**</b> were passed to the <b>**MemSpy::finishMemoryMonitoring(...)**</b> function, at the end of the example program, the memory leaks are displayed in the console and in the VisualStudio output window.

\endif

\if german

Da der Funktion <b>**MemSpy::finishMemoryMonitoring(...)**</b> die Parameter <b>**MemSpy::OUTPUT\_TRACE**</b> und <b>**MemSpy::OUTPUT\_CONSOLE**</b> übergeben wurden, werden am Ende des Beispielprogramms die Memory-Leaks in der Konsole und im VisualStudio Ausgabefenster angezeigt.

\endif

\code

Der Thread 15572 hat mit Code 0 (0x0) geendet.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Memory leaks detected! \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Memory leak in 1313691209200, file D:\Projekte\C++\Src\MemoryLeakIndicator\source\main.cpp, line 45

\* Memory leak in 1313691165808, file D:\Projekte\C++\Src\MemoryLeakIndicator\source\main.cpp, line 21

\* Memory leak in 1313691166000, file D:\Projekte\C++\Src\MemoryLeakIndicator\source\main.cpp, line 48

\* Memory leak in 1313691152224, file D:\Projekte\C++\Src\MemoryLeakIndicator\source\main.cpp, line 51

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"MemoryLeakIndicatorDllTestd.exe" (Win32): "C:\Windows\System32\kernel.appcore.dll" geladen.

"MemoryLeakIndicatorDllTestd.exe" (Win32): "C:\Windows\System32\msvcrt.dll" geladen.

"MemoryLeakIndicatorDllTestd.exe" (Win32): "C:\Windows\System32\rpcrt4.dll" geladen.

Der Thread 10624 hat mit Code 0 (0x0) geendet.

Der Thread 17076 hat mit Code 0 (0x0) geendet.

Detected memory leaks!

Dumping objects ->

{172} normal block at 0x00000131DE182F60, 100 bytes long.

Data: < > CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD

{171} normal block at 0x00000131DE186530, 4 bytes long.

Data: < > 02 00 00 00

{170} normal block at 0x00000131DE186470, 4 bytes long.

Data: < > 01 00 00 00

{169} normal block at 0x00000131DE190DF0, 8 bytes long.

Data: <pd 1 > 70 64 18 DE 31 01 00 00

Object dump complete.

Das Programm "[15928] MemoryLeakIndicatorDllTestd.exe" wurde mit Code 0 (0x0) beendet.

\endcode

\if english

**\note** If you create objects in modern C++ projects and pass them on if necessary, you do not use simple pointers if possible, but smart pointers like <b>**std::unique\_ptr**</b> or <b>**std::shared\_ptr</**b>.As a rule, the specifications of RAII should be followed.

**\sa** https://docs.microsoft.com/en-us/cpp/cpp/object-lifetime-and-resource-management-modern-cpp?view=msvc-170 <br>

https://docs.microsoft.com/en-us/cpp/cpp/how-to-create-and-use-shared-ptr-instances?view=msvc-170

\endif

\if german

**\note** Wenn man in modernen C++ Projekten Objekte erzeugt und ggf. weiterreicht, verwendet man möglichst keine einfachen Zeiger, sonder Smart-Pointer wie <b>**std::unique\_ptr**</b> oder <b>**std::shared\_ptr</**b>. Es sollte sich i.d.R. nach den Vorgaben von RAII gerichtet werden.

**\sa** <https://docs.microsoft.com/de-de/cpp/cpp/object-lifetime-and-resource-management-modern-cpp?view=msvc-170> <br>

https://docs.microsoft.com/de-de/cpp/cpp/how-to-create-and-use-shared-ptr-instances?view=msvc-170

\endif

\if english

# \section **MLI\_LIMITATIONS Limitations**

I failed to overload the <b>**new (std::nothrow)**</b>. When dealing with custom code, this new operator must either be modified for this purpose (without std::nothrow), or <b>**MilDebugNew.h**</b> is not included.

<br>

Every now and then, especially in multi-threaded applications, the <b>**MemoryLeakIndicator**</b> shows <i>*fake*</i> memory leaks. These are recognizable if you run the application in debug mode using <b>**crtdbg.h**</b> and there is no additional output in the output in the VisualStudio output window such as:

\code

…

Detected memory leaks!

Dumping objects →

{172} normal block at 0x00000131DE182F60, 100 bytes long.

Data: < > CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD

...

\endcode

<br>

Trials with <b>**MinGW**</b> and <b>**GCC**</b> have also failed for me; only works with <b>**VisualStudio C++**</b>. If anyone is willing to adapt <b>**MemoryLeakIndicator**</b> to other compilers, I'd appreciate it.

\endif

\if german

# \section **MLI\_LIMITATIONS Einschränkungen**

Mir gelang es nicht, den <b>**new (std::nothrow)**</b> zu überladen. Wenn es sich um eigenen Code handelt, muss dieser new operator entweder für diesen Zweck geändert werden (ohne std::nothrow), oder es wird <b>**MilDebugNew.h**</b> nicht eingebunden.

<br>

Ab und an, gerade in Multithread-Anwendungen, werden vom <b>**MemoryLeakIndicator**</b> <i>*unechte*</i> Memory-Leaks angezeigt. Diese sind erkennbar, wenn man die Anwendung im Debug-Modus unter Verwendung von <b>**crtdbg.h**</b> laufen lässt und in der Ausgabe im VisualStudio Ausgabefenster keine zusätzlich Ausgabe gibt wie z.B.:

\code

…

Detected memory leaks!

Dumping objects →

{172} normal block at 0x00000131DE182F60, 100 bytes long.

Data: < > CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD CD

...

\endcode

<br>

Versuche mit <b>**MinGW**</b> und <b>**GCC**</b> sind bei mir auch fehlgeschlagen; geht also nur mit <b>**VisualStudio C++**</b>. Wenn jemand bereit ist, <b>**MemoryLeakIndicator**</b> an andere Compiler anzupassen, würde ich mich freuen.

\endif

**\*/**