31 August 2017

VS2017 + modules

- Modules + `module` identifier => failed build
- Compiler: `msvc140` vs. `msvc141`
- Boost: no more `std::unary_function` and `std::binary_function` (fixed in 1.64)
- IncrediBuild support for modules: ???

How to use the C++ Core Guidelines Checker outside of Visual Studio

Post

- VS2017.3

```
<PropertyGroup>
  <EnableCppCoreCheck>true</EnableCppCoreCheck>
  <CodeAnalysisRuleSet>
    CppCoreCheckRules.ruleset
  </CodeAnalysisRuleSet>
  <RunCodeAnalysis>true</RunCodeAnalysis>
</PropertyGroup>
```

```
msbuild /p:EnableCppCoreCheck=true
/p:RunCodeAnalysis=true
/p:CodeAnalysisRuleSet=CppCoreCheckRules.ruleset ...
```

Software development 450 words per minute

- Post
  - Uses Notepad++ and Intellij IDEA
  - Sublime Text: not accessible
- NVDA screen reader – open-source, Windows-only
- VoiceOver – Mac and iOS
- Orca – Gnome
- How a blind person uses Visual Studio – YouTube

Functional data structures in C++ by Bartosz Milewski, C++Now 2014

- YouTube
- GitHub
- Blog post
- C++ (naive implementation) vs. Haskell is 1000x slower:
  - heap allocation: big blobs vs. small chunks
  - smart pointers vs. GC
  - native (non-pooled) threads vs. lightweight threads
  - sub-optimal synchronisation
  - not using move semantics properly
  - `std::function` allocates

David Stone - Writing Robust Code - Meeting C++ 2014

YouTube

- Story

MTuner

GitHub

MTuner is a C/C++ memory profiler and memory leak finder for Windows, PS4, PS3, etc.

MTuner utilizes a novel approach to memory profiling and analysis, keeping entire time-based history of memory operations. This gives an unique insight in memory related behavior of your software by making queries over the entire data set.

Licence: BSD-2

Post

- Requires VS 2017 15.4

pybind11

GitHub

- Seamless operability between C++11 and Python
- Header only
- Exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code
- Similar to Boost.Python library

C++17: file system

- Post by Bartlomiej (Bartek) Filipek
- Reddit thread

C++ documentation generators

- Reddit thread
- Doxygen (sample)
- Doxypress
- Doxygen + Sphinx + Breathe (sample1, sample2)
- Standardese (sample)
- Robert Ramey, How to Write Effective Documentation for C++ Libraries with Minimal Effort, CppCon 2017

C++/WinRT

- Modern C++ Blog by Kenny Kerr
- GitHub
- C++17
- Coroutines
- Clang support
- `[[deprecated]]`
- Simplified ABI interop
- “Optimized agility implementation”??? `IAgileObject` :-|

libnyquist: audio decoding library

- GitHub
- C++11
- Ogg Vorbis, FLAC, Wave, MIDI, MOD etc.
- BSD licence

FlaggedT: type level flagging

`Flagged<T>` offers multiple wrapper types which allow you to add properties to your variables at type level. The wrapped types can still be used as the inner type, thanks to operator overloading.

- GitHub
- MIT licence
- Single header

Top 20 C++ multithreading mistakes and how to avoid them

- Post
- Reddit thread

  Honestly disappointed that list was in order. What a missed opportunity.

HexType: Efficient Detection of Type Confusion Errors for C++

PDF

- Run-time defences against type confusion errors (bad casts)
- Less run-time overhead and better detection than TypeSan, CaVer, UBSan
- LLVM-based

Custom type traits

Post

```
// only WANTED type available
template<typename WANTED, typename...>
struct can_be_used_as : std::true_type
{};

// Processes all given types
template<typename WANTED,
         typename HEAD,
         typename... TAIL>
struct can_be_used_as<WANTED, HEAD, TAIL...>
    : std::integral_constant<
        bool,
        (std::is_same<WANTED,HEAD>{} ||
        std::is_base_of<WANTED,HEAD>{} ||
        std::is_convertible<HEAD,WANTED>{}) &&
        can_be_used_as<WANTED, TAIL...>{}>
{};
```

Simplify your type traits with C++14 variable templates

Post

Before:

```
template <typename T>
struct is_float {
    static constexpr bool value =
        std::is_same<T, float>::value;
};
```

or

```
template <typename T>
using is_float =
std::integral_constant<bool,
                       std::is_same<T, float>::value>;
```

Usage:

```
template <typename T>
void test(T t){
    if (is_float<T>::value){
```

Metabench: compile-time benchmarks

- Author: Louis Dionne
- GitHub
- A single CMake module
- Requires CMake 3.1 and Ruby 2.1
- Works with CMake and Ninja
- Boost licence

Ignore std::bad_alloc?

Reddit

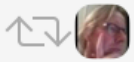
On Windows, thrown exceptions are copied onto the stack. Destructors and catch blocks are “called” with all of the previous function frames still on the stack. Only when the exception is handled is the stack pointer popped. This has the unfortunate side effect that rethrowing an exception in a deep call stack can cause stack exhaustion.

With the Itanium ABI (Linux, Mac, etc.), the exception is pretty much forced to be on the heap. The compiler will emit calls to `void *__cxa_allocate_exception(size_t)` and `void __cxa_free_exception(void *)`. However, the implementations of those functions aren’t required to use `new`. The LLVM implementation uses `malloc`, and if that fails, it goes to an emergency fallback store of 512 bytes.

If you’re running on Linux then all memory allocations where the size is less than the physical ram succeed because it uses overcommit. All of them, whether there’s enough available memory left or not. Your application will just crash when you try to use it.

Two chevrons

Chris Oldwood @chrisoldwood
I remember when the same advice applied to C++ template type lists. pic.twitter.com/zDY31q7Twa


Björn Fahller