Tales from C++ On Sea

Martyn Gigg
Mantid Developer Workshop - 1st April 2019

C++ On Sea

- 2 day conference (+ 1 day workshop) in Folkestone, UK
- Purely C++
- Dan will lead us through an exercise he did in the workshop later

Videos at https://www.youtube.com/channel/UCAczr0j6ZuiVaiGFZ4qxApw

Practical Performance Practices (revisited)

- Predicting what the compiler can optimize is hard
- Containers:
    - Prefer `std::array, std::vector` but always measure!
- Always const => always initialise when const not practical
- Constexpr where you can
- Move static data to constexpr
- Prefer '\n' to std::endl
- Avoid meaningless destructors => compiler can inline stuff if you don't get in the way
- Use `shared_ptr/map/deque` sparingly

C++17: New, Fixed & Pitfalls

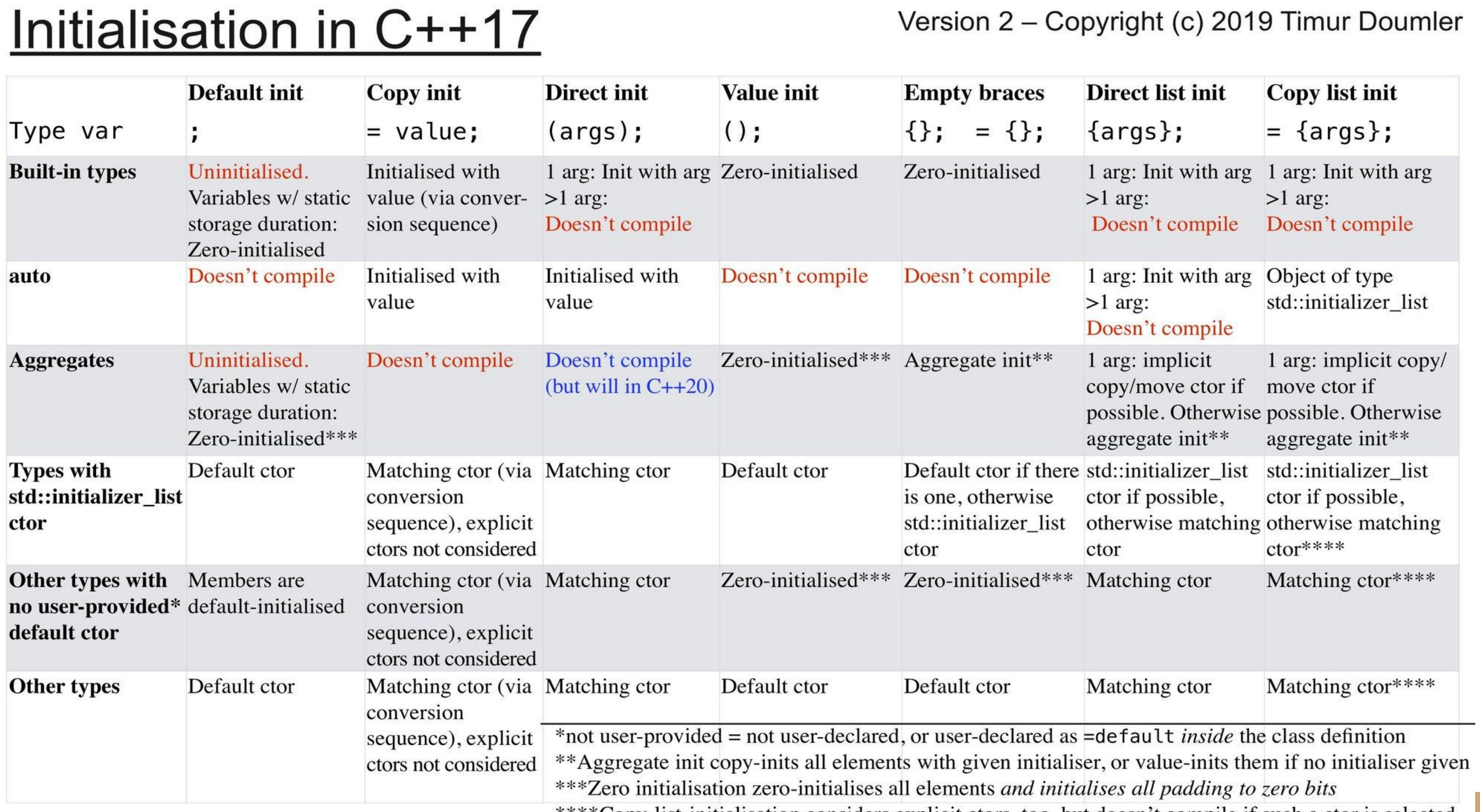
- Advice is to prefer uniform initialisation syntax `{}`
  - C++11 has a 'quirk':
    - `auto x{42};` # x is an int
    - `auto x = {42};` # x is an initializer_list!
    - fixed in C++17
- Init conditions in `if/switch`
  - `if (auto it = m.find(10); it != m.end()) { return it->second.size(); }`
  - Name is valid for scope of `if`
  - Similar syntax for `switch`
- Defined expression evaluation order
  - cout << f() << g() << h()
  - fixed in C++17 for some operators
  - Note that argument evaluation order is still undefined, e.g f(a,b,c) undefined as to order of a,b,c

C++17: New, Fixed & Pitfalls

- Class Template Argument Deduction (CTAD)
  - std::lock_guard(mtx); => No <> required!
- Library:
  - std::optional
    - std::optional<bool> ob{false};
    - if(!ob) // false as ob has a value
    - if(ob == false) // true as the value is false
    - .value() throws is no value
    - deference operator *ob does not check
  - std::string_view
    - Handle for read only strings
    - Don’t use as return type
  - std::filesystem
    - Differences to version in boost r.e. handling of “$HOME/.git”
  - Polymorphic memory resources
    - Interface for allocators that don’t form part of the type

Modern C++ Initialization:

https://twitter.com/timur_audio/status/1096101040200581122



Initialisation in C++17

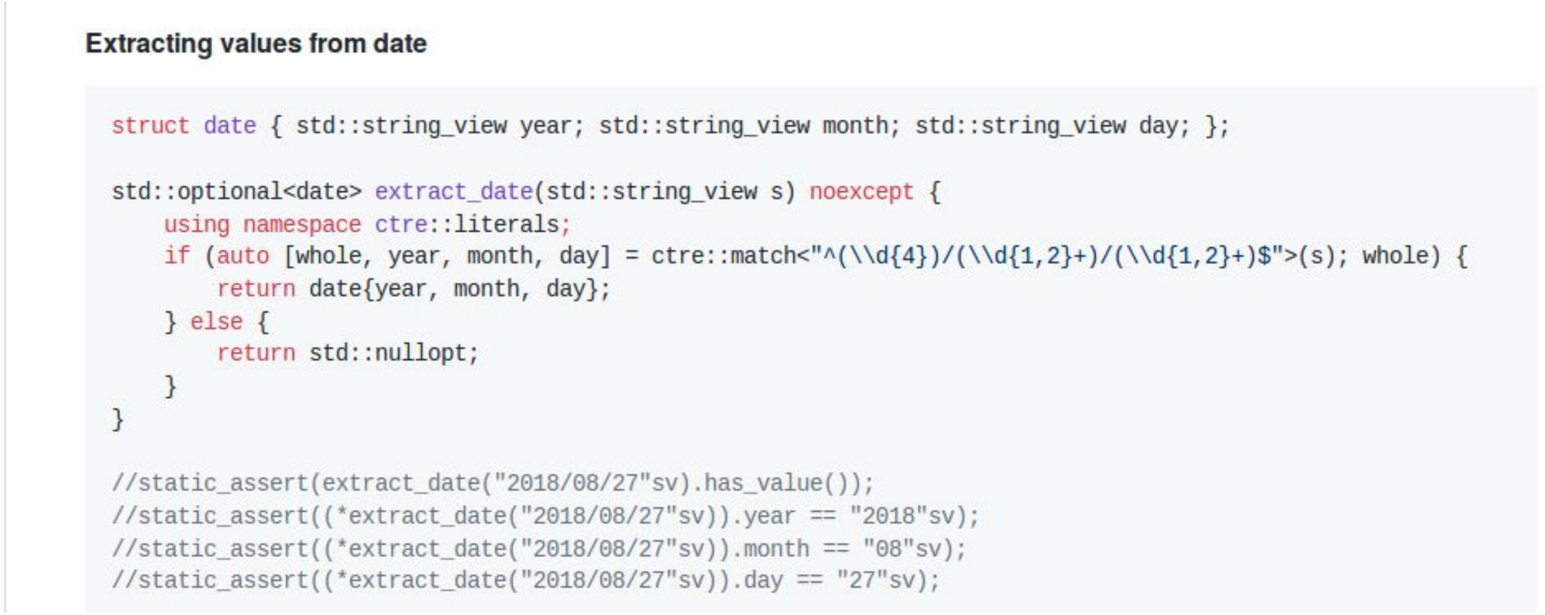
Version 2 – Copyright (c) 2019 Timur Doumler

`Type var`	**Default init** `;`	**Copy init** `= value;`	**Direct init** `(args);`	**Value init** `();`	**Empty braces** `{};  = {};`	**Direct list init** `{args};`	**Copy list init** `= {args};`
Built-in types	Uninitialised. Variables w/ static storage duration: Zero-initialised	Initialised with value (via conver-sion sequence)	1 arg: Init with arg >1 arg: Doesn't compile	Zero-initialised	Zero-initialised	1 arg: Init with arg >1 arg: Doesn't compile	1 arg: Init with arg >1 arg: Doesn't compile
auto	Doesn't compile	Initialised with value	Initialised with value	Doesn't compile	Doesn't compile	1 arg: Init with arg >1 arg: Doesn't compile	Object of type std::initializer_list
Aggregates	Uninitialised. Variables w/ static storage duration: Zero-initialised***	Doesn't compile	Doesn't compile (but will in C++20)	Zero-initialised***	Aggregate init**	1 arg: implicit copy/move ctor if possible. Otherwise aggregate init**	1 arg: implicit copy/ move ctor if possible. Otherwise aggregate init**
Types with std::initializer_list ctor	Default ctor	Matching ctor (via conversion sequence), explicit ctors not considered	Matching ctor	Default ctor	Default ctor if there is one, otherwise std::initializer_list ctor	std::initializer_list ctor if possible, otherwise matching ctor	std::initializer_list ctor if possible, otherwise matching ctor****
Other types with no user-provided* default ctor	Members are default-initialised	Matching ctor (via conversion sequence), explicit ctors not considered	Matching ctor	Zero-initialised***	Zero-initialised***	Matching ctor	Matching ctor****
Other types	Default ctor	Matching ctor (via conversion sequence), explicit ctors not considered	Matching ctor	Default ctor	Default ctor	Matching ctor	Matching ctor****

*not user-provided = not user-declared, or user-declared as `=default` *inside* the class definition
**Aggregate init copy-inits all elements with given initialiser, or value-inits them if no initialiser given
***Zero initialisation zero-initialises all elements *and initialises all padding to zero bits*
****Copy list initialisation considers explicit ctors, too, but doesn't compile if such a ctor is selected

Compile-time Regular Expressions

- https://github.com/hanickadot/compile-time-regular-expressions



Extracting values from date

```
struct date { std::string_view year; std::string_view month; std::string_view day; };

std::optional<date> extract_date(std::string_view s) noexcept {
    using namespace ctre::literals;
    if (auto [whole, year, month, day] = ctre::match<"^(\\d{4})/(\\d{1,2}+)/(\\d{1,2}+)$">(s); whole) {
        return date{year, month, day};
    } else {
        return std::nullopt;
    }
}

//static_assert(extract_date("2018/08/27"sv).has_value());
//static_assert((*extract_date("2018/08/27"sv)).year == "2018"sv);
//static_assert((*extract_date("2018/08/27"sv)).month == "08"sv);
//static_assert((*extract_date("2018/08/27"sv)).day == "27"sv);
```

Other things

- Nice keynote from Matt Godbolt:
    - What Everyone Should Know About How Amazing Compilers Are:
        - https://www.youtube.com/watch?v=w0sz5WbS5AM
- C++20
    - Talk on Contracts: https://www.youtube.com/watch?v=Dzk1frUXq10
    - It’s looking like another big release. Design finalized, waiting for wording completion at next meeting
        - https://www.reddit.com/r/cpp/comments/au0c4x/201902_kona_iso_c_committee_trip_report_c20/
        - Ranges!
        - Modules
        - Coroutines
        - Concepts
        - std::format!
        - operator <=>
        - ...

Radnor Arms