Getting type name at compile time

Adam Badura, Nokia

We have the function name

- `__func__` (C99, C++11)
- `__PRETTY_FUNCTION__` (GCC & clang)
- `__FUNCSIG__` (MSVC)
- `BOOST_CURRENT_FUNCTION`

What about the class name?

- Is there a __CLASS__ macro in C++? - StackOverflow
- Preprocessor macro to get the name of the current class? [duplicate] - StackOverflow
- Is there a class name macro? - Bytes
- Is there a __CLASS__ macro in C++? - CODE A&A Solved

`type_name` to the rescue!

```
static_assert(type_name_v<int> == "int");
static_assert(type_name_v<decltype(0.1 * 10)> == "double");
```

With the help of `__PRETTY_FUNCTION__` / `__FUNCSIG__`

```
template<typename T>
void foo()
{
        std::puts(__PRETTY_FUNCTION__); // for GCC & clang
        //std::puts(__FUNCSIG__); // for MSVC
}
```

- GCC: `void foo() [with T = {type}]`
- clang: `void foo() [T = {type}]`
- MSVC: `void __cdecl foo<{type}>(void)`

Standard `__func__` is useless here

```
template<typename T>
void foo()
{
        std::puts(__func__);
}
```

- GCC: `foo`
- clang: `foo`
- MSVC: `foo`

Underlying type

- All behave as if they were string literals.
- Much like `__FILE__`.
- However, those are not preprocessor symbols and will outlive preprocessing phase!

`constexpr` and `std::string_view`

```
template<typename T>
constexpr auto foo()
{
        constexpr std::string_view full_name{ __PRETTY_FUNCTION__ };
        constexpr std::string_view left_marker{ "[with T = " };
        constexpr std::string_view right_marker{ "]" };

        constexpr auto left_marker_index = full_name.find(left_marker);
        static_assert(left_marker_index != std::string_view::npos);
        constexpr auto start_index = left_marker_index + left_marker.size();
        constexpr auto end_index = full_name.find(right_marker, left_marker_index);
        static_assert(end_index != std::string_view::npos);
        constexpr auto length = end_index - start_index;

        return full_name.substr(start_index, length);
}
```

Tricky C-printing! 😟

```
std::cout << type_name_v<int>;
```

or

```
constexpr auto name = type_name_v<int>;
std::printf("%.*s\n", static_cast<int>(name.size()), name.data());
```

Unaware of aliases 😞

```
static_assert(type_name_v<std::size_t> == "long unsigned int");
```

Compiler dependent 😞

- GCC

```
static_assert(type_name_v<std::string> == "std::__cxx11::basic_string<char>");
```

- clang

```
static_assert(type_name_v<std::string> == "std::__cxx11::basic_string<char, std::char_tr
```

GitHub

https://github.com/adambadura/type_name

Compiler Explorer

https://godbolt.org/z/vaPf7l

Q&A