Document Number: N3987
Date: 2014-05-07
Project: SG7 - Reflection
Reply to: Cleiton Santoia Silva
<cleitonsantoia@gmail.com>
Daniel Auresco
<auresco@gmail.com>

Yet another set of C++ type traits.

Contents

Contents **ii**

1 Proposal **1**

1.1 Intro 1

1.2 Motivation 1

1.3 Design 1

Bibliography **3**

1 Proposal [prop]

1.1 Intro [prop.intro]

In C++, types can carry a big amount of extra information, by attaching many language constructs like virtual, inline, delete and so on. But without the corresponding type traits.

1.2 Motivation [prop.motivation]

The content of this paper was originally part of [SA14, N3951], the design of C++ static type reflection need a way to retrieve important information about class members, for example, if it is public/protected/private, so, after some thought the simplest way: just put `is_` before public/protected/private keywords to check if that instruction appear in the member specification. But since the implementation of it do not depend at all of [SA14, N3951], we choose to put this in a different paper.

1.3 Design [prop.design]

Just as exercise you can pick all keywords of C++ and put a `is_` before and thought if it make sense. Many of them are just check if in the declaration of the parameter type have a particular keyword and others just check if some constructs, but any . This way, all type traits we propose here have boolean answers.

— `std::is_default<F>` returns true if member function F is marked as default.

— `std::is_delete<F>` returns true if function F is marked as delete.

— `std::is_extern<D>` returns true if declaration D is marked as extern.

— `std::is_explicit<F>` returns true if function F is marked as explicit.

— `std::is_export<D>` return true if declaration D is marked as export.

— `std::is_final<D>` return true if declaration D is marked as final.

— `std::is_friend<T,U>` return true if type T is declared as friend of U.

— `std::is_inline<F>` return true if function F is marked as inline.

— `std::is_mutable<M>` return true if member M is marked as mutable.

— `std::is_noexcept<F>` return true if function F is marked as noexcept.

— `std::is_override<F>` return true if function F is marked as override.

— `std::is_private<M>` return true if member M is inside a private scope.

— `std::is_private_base_of<Base,Derived>` return true if class “Base” is a private base class of “Derived”.

— `std::is_protected<M>` return true if member M is inside a protected scope.

— `std::is_protected_base_of<Base,Derived>` return true if class “Base” is a protected base class of “Derived”.

— `std::is_public<M>` return true if member M is inside a public scope.

— `std::is_public_base_of<Base,Derived>` return true if class “Base” is a public base class of “Derived”.

— `std::is_thread_local<V>` return true if variable V is has a thread local defined.

— `std::is_virtual<F>` return true if function F is virtual.

— `std::is_virtual_base_of<Base,Derived>` return true if class “Base” is a virtual base class of “Derived”.

Another traits that are not defined directly by the presence of keywords, but from some instruction construct.

— `std::is_local<T>` return true if the class is a local class (defined inside a function).

— `std::is_pure_virtual<F>` return true if the function F is a pure virtual function.

— `std::is_overload<F>` return true if function F has some overload.

— `std::is_direct_base_of<Base,Derived>` return true if class “Base” is a direct base class of “Derived”. Inside [Spe09, N2965] is defined a trait that can get same information by retrieving a typelist of base classes, and also TR2 we have `std::tr2::direct_bases<Foo>::type` trait to get a typelist of base classes.

As expected, all of them must be declared inside `<type_traits>` header.
We checked a little if proposed type traits was previously proposed, so please if someone found some of these in some other paper, please let us know and we will remove from this paper and cite the previous paper.

Bibliography

[SA14] Cleiton Santoia Silva and Daniel Auresco. N3951 - c++ type reflection via variadic template expansion. Technical report, C++ SG7, 2014.

[Spe09] Michael Spertus. N2965 - type traits and base classes. Technical report, Symantec, 2009.