Recursive Type Template Instantiation

Document number:	P1230R0
Date:	2018-10-03
Project:	Programming Language C++
Audience:	EWG, CWG
Authors:	Mingxin Wang
Reply-to:	Mingxin Wang <wmx16835vv@163.com>

Table of Contents

1 Introduction

Currently, we are not able to refer to a type in its template. For example, it is impossible to let the return type of an instantiation of **`std::function`** to be itself: **`std::function<std::function<std::function< /* infinite recursion */ >()>()>`**. And we found it a common requirement in polymorphic programming.

Here is a use case in our production: there are many different periodic tasks in our system, some execute only once, most of them execute several times with different and variable intervals. Therefore, it is required to design a runtime abstraction for periodic task type to submit to one generic "timed thread pool" that manages several threads. In order to avoid unnecessary "submission" that may introduce overhead in contention (e.g., **`acquire`** & **`release`** some mutex), we think it could be efficient to let the execution of the abstraction output a type of itself, e.g.: **`std::function<std::optional<std::pair<std::chrono::time_point<std::chrono::system_clock>, std::function< /* infinite recursion */ >>>()>`**, but we are not able to define that type within finite characters, and it turned out that we cannot use std::function in this case.

Therefore, we propose a new syntax for type template to refer to a part of itself, recursively. The syntax is to use the template name as a recursive typename. For example, if we have **`template <class> class A`** and **`template <class> class B`**, then **`A<B<A>>`** has the same semantics of **`A<B<A<B< /* infinite recursion */ >>>>`**.

2 Technical Specification

We propose the following content to be merged into clause 12.8 [**`temp.spec`**]:

A *template-name* could be used as a *template-argument* as a "*recursive-template-argument*" when the corresponding *template-parameter* is a *typedef-name*.

The *template-name* of a *recursive-template-argument* shall appear in its upper-level template instantiation. [*Note*: For example, while **`A<B<A>>`** is well-formed, **`A<B<C>>`** is not, providing **`C`** is a type template. — *end note*]

A *recursive-template-argument* shall be replaced into its nearest upper-level template instantiation of the same template-name during its template instantiation. [*Note*: For example, during the instantiation of **`A<B<A>>`**, the template **`B`** shall be instantiated as **`B<A<B<A>>>`**. — *end note*]

[*Note*: There is no specific rule to determine the equivalence among template instantiations with *recursive-template-arguments*. For example, **`A<B<A>>`** and **`A<B<A<B<A>>>>`** are not the same type, even if they generate similar code. However, the use of **`A<B<A<B<A>>>>`** could usually be reconstructed with **`A<B<A>>`**, and thus compilers may warn for such usage. — *end note*]

3 Known Use Case

The "timed thread pool" mentioned in the "Introduction" part was implemented with a hand-written recursive template implementation, whose runtime abstraction is supported by the PFA (P0957).