Allowing multiple template parameter packs for class templates provided they can be deduced using CTAD

Document #: DXXXXR0
Date: 2023-04-29
Project: Programming Language C++
Audience: Evolution Working Group
Reply-to: Anoop Rana
<ranaanoop986@gmail.com>

1 Introduction

This paper proposes that class templates be allowed to have multiple template parameter packs as long as they can be deduced using class template argument deduction(CTAD).

Currently the standard does not allow the use of multiple template parameter packs in class templates. But since C++17 added CTAD, it should be possible to have multiple template parameter packs in a class template **as long as** those can be deduced using CTAD. This is like multiple template parameter packs are already allowed by the standard in "function templates" as long they can be deduced.

Below is an example demonstrating the issue:

```cpp
//this is allowed and works as expected using deduction
template< typename... Param,typename Ret, typename... Args>
void doStuff(Ret (*ptrFunc)(const Param&... param),const Args&... args)
{
}
/*this is not allowed according to the current wording but should
be allowed from C++17(&onwards) since we have CTAD*/
template< typename... Param,typename Ret, typename... Args>
struct C
{
    C(Ret (*ptrFunc)(const Param&... param),const Args&... args);
};
```

2 Motivation and Scope

The main motivation is to increase consistency of CTAD with normal template argument deduction that happens with function templates with multiple template parameter packs where they can be deduced. Additionally, this will increase usability of class templates.

3 Impact on the Standard

This proposal will only make currently invalid programs(that have multiple template parameter packs that can be deduced using CTAD) valid. That is, currently accepted program will not be affected/rejected by the introduction of this proposal into the standard.

4 Before-After Comparision(s)

4.1 Before

```
//ill-formed
template< typename... Param,typename Ret, typename... Args>
struct C
{
    C(Ret (*ptrFunc)(const Param&... param),const Args&... args);
};
```

4.2 After

```
//well-formed
template< typename... Param,typename Ret, typename... Args>
struct C
{
    C(Ret (*ptrFunc)(const Param&... param),const Args&... args);
};
```