Wording for class template argument deduction from inherited constructors

Timur Doumler (papers@timur.audio)

Document #: P2582R0
Date: 2022-05-15
Project: Programming Language C++
Audience: Core Working Group

Abstract

This paper provides wording for class template argument deduction from inherited constructors [P1021R6].

1 Proposed wording

The proposed changes are relative to the C++ working draft [N4910].

In [over.match.class.deduct], append to paragraph 1 as follows:

> except that additional parameter packs of the form `P`$_j$`...` are inserted into the parameter list in their original aggregate element position corresponding to each non-trailing aggregate element of type `P`$_j$ that was skipped because it was a parameter pack, and the trailing sequence of parameters corresponding to a trailing aggregate element that is a pack expansion (if any) is replaced by a single parameter of the form `T`$_n$`...`.
>
> In addition, if `C` inherits constructors (namespace.udecl) from a base class denoted in the *base-specifier-list* by a *simple-template-id* `B`, the set contains the functions and function templates formed from an alias template whose template parameters are those of `C` and whose *simple-template-id* is `B`.

In [over.match.class.deduct], add the following example to the existing block of examples:

> [*Example:*
>
> ```
> template <typename T> struct Base {
>   Base(T&&);
> };
>
> template <typename T> struct Derived : public Base<T> {
>   using Base<T>::Base;
> }
> ```

```
Derived d(42);   // OK, deduces Derived<int>
```

— *end example*]

In [over.match.best.general], insert as follows:

— `F1` and `F2` are rewritten candidates, and `F2` is a synthesized candidate with reversed order of parameters and `F1` is not [*Example:*

```
struct S {
  friend std::weak_ordering operator<=>(const S&, int);          // #1
  friend std::weak_ordering operator<=>(int, const S&);          // #2
};
bool b = 1 < S();                                                // calls #2
```

— *end example*] or, if not that,

— `F1` is generated from class template argument deduction ([over.match.class.deduct]) for a class `D`, `F2` is generated from inheriting constructors from a base class of `D`, and for all arguments the corresponding parameters of `F1` and `F2` have the same type, or, if not that,

— `F1` is generated from a *deduction-guide* ([over.match.class.deduct]) and `F2` is not, or, if not that,

References

[N4910] Thomas Köppe. Working Draft, Standard for Programming Language C++. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/n4910.pdf, 2022-03-17.

[P1021R6] Mike Spertus, Timur Doumler, and Richard Smith. Filling holes in Class Template Argument Deduction. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p1021r6.html, 2022-05-15.