Wording for class template argument deduction from inherited constructors

Timur Doumler (papers@timur.audio)

Abstract

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

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 \( P_j \ldots \) 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 \( T_n \ldots \).

In addition, if \( C \) is defined and inherits constructors ([namespace.udecl]) from a direct base class denoted in the base-specifier-list by a class-or-decltype \( B \), let \( A \) be an alias template whose template parameter list is that of \( C \) and whose defining-type-id is \( B \). If \( A \) is a deducible template ([dcl.type.simple]), the set contains the guides of \( A \) with the return type \( R \) of each guide replaced with typename CC<R>::type given a class template

\[
\text{template<typename} CC; \\
\]

whose primary template is not defined and with a single partial specialization whose template parameter list is that of \( A \) and whose template argument list is a specialization of \( A \) with the template argument list of \( A \) ([temp.dep.type]) having a member typedef type designating a template specialization with the template argument list of \( A \) but with \( C \) as the template.

[Note: Equivalently, the template parameter list of the specialization is that of \( C \), the template argument list of the specialization is \( B \), and the member typedef names \( C \) with the template argument list of \( C \). — end note]
In [over.match.class.deduct], add the following example:

```cpp
Example:

template <typename T> struct B {
    B(T);
};

template <typename T> struct C : public B<T> {
    using B<T>::B;
};

template <typename T> struct D : public B<T> {};
C c(42); // OK, deduces C<int>
D d(42); // Error: deduction failed, no inherited deduction guides
B(int) -> B<char>;
C c2(42); // OK, deduces C<char>

template <typename T> struct E : public B<int> {
    using B<int>::B;
};
E e(42); // Error: deduction failed, arguments of E cannot be deduced from guides introduced

template <typename T, typename U, typename V> struct F {
    F(T, U, V);
};

template <typename T, typename U> struct G : F<U, T, int> {
    using G::F::F;
}
G g(true, 'a', 1); // OK, deduces G<char, bool>
— 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:

```cpp
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 and F2 are generated from class template argument deduction ([over.match.class.deduct]) for a class D, and F2 is generated from inheriting constructors from a base class of D while F1 is not, and for each explicit function argument, the corresponding parameters of F1 and F2 are either both ellipses or 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,
2 Known issues

The mechanism for class template argument deduction from inherited constructors proposed here relies on the existing mechanism for class template argument deduction from alias templates. Core issue [CWG2467] should be expanded to include additional instances of the problem introduced by this paper.

Document history

— R1, 2022-05-20: Wording changes following CWG review.

Acknowledgements

Many thanks to Hubert Tong for his help with fixing the wording.

References

