Abstract

This document constitutes a short remark to P1170: Overload sets as function parameters and P0051: C++ generic overload function. From a library design standpoint we argue that the functionality provided by these papers should be considered together, and not independently. In particular, we feel that the current form of P1170 will not solve the problem of variadic overload sets built from non-inheritable classes that P0051 is facing. Making std::overload_set a variadic template class would allow to solve this problem.
Proposal

Before

1 // From P1170
2 template <typename T>
3 class overload_set;
4
5 // From P0051
6 template <class ... Fs>
7 auto overload(Fs&&... fcts);
8
9 // From P0051
10 template <class ... Fs>
11 auto /first_overload(Fs&&... fcts);

After

1 // A variadic overload set: calls the best callable
2 template <class... F>
3 class overload_set;
4
5 // A variadic overload set: calls the first callable that works
6 template <class... F>
7 class overload_sequence;

First, instead of one class, we introduce two: one that calls the best callable according to overload rules, and the other one that calls the first callable that works. We think that the symmetry of names overload_set/overload_sequence provides clarity for the user. Second, the suggested implementation of the non-variadic overload_set in P1170 is:

Suggested implementation of non-variadic overload_set in P1170

1 template <typename T>
2 class overload_set {
3  T f;
4 public:
5  overload_set(/* unspecified */);
6  overload_set(overload_set const&) = default;
7  overload_set(overload_set&&) = default;
8  overload_set& operator=(overload_set const&) = default;
9  overload_set& operator=(overload_set&&) = default;
10  ~overload_set() = default;
11
12  template <typename... Us>
13  invoke_result_t<F&, Us...> operator()(Us&&... us) &
14  noexcept(is_nothrow_invocable_v<F&, Us...>);
15
16  template <typename... Us>
17  invoke_result_t<F const&, Us...> operator()(Us&&... us) const &
18  noexcept(is_nothrow_invocable_v<F const&, Us...>);
19
20  template <typename... Us>
21  invoke_result_t<F, Us...> operator()(Us&&... us) &&
22  noexcept(is_nothrow_invocable_v<F, Us...>);
23
24  template <typename... Us>
25  invoke_result_t<F const, Us...> operator()(Us&&... us) const&&
26  noexcept(is_nothrow_invocable_v<F const, Us...>);
27  
28  }

With this definition, a template class inheriting from two overload_set to combine two overload sets with different names may not work properly. Introducing an overload_set which does not provide a solution to this problem is
very likely to be a dealbreaker for users. If overload_set is meant to be a “compiler magic” class anyway, it should be variadic and correctly handle the union of several overload sets.

At this point, we do not suggest a specific solution. Our goal was to highlight this potential problem to foster interest. This paper will be updated (potentially with a solution) before the Cologne 2019 Standards Committee Meeting.

References


- **Custom Overload Sets and Inline SFINAE for Truly Generic Interfaces**, Vincent Reverdy, *CppCon 2018* (September 2018)