P2669R0

Deprecate changing kind in class template specialization

Bengt Gustafsson

Kona - 2022

Presentation contents

- Rationale
- Proposal contents

Rationale

- Changing kind of a name between definition and explicit specialization is most often a bug.
- If parsing can rely on the kind in the definition we can get rid of a majority of all disambiguation using **typename** and **template**.
- The reasons for not having get<> as a member function in tuple etc. are thereby removed.

Proposal contents

- Deprecate changing kind of names in one revision (C++23?)
- Forbid kind change and allow compilers to rely on kind in class template definition during parsing in the next revision.

Alternatives

- Just ask compiler vendors to issue warnings ASAP.
- Forbid kind change directly in C++26.

Alternative approach (not in P2669R0)

- Don't forbid changing kind as such.
- Still let compiler use kind from definition when parsing unless disambiguated to something else.
- Needs a way to disambiguate to **value**.
- More confusion risk when errors are to be diagnosed.

Examples

```
#include<vector>

template<typename T> void foo(std::vector<T> v)
{
    typename std::vector<T>::iterator iter = v.begin();    // Why! Boring!
}


template<typename T> void printIter()
{
    std::cout << valuename vector<T>::iterator;  // Alternative: disambiguate when expecting other kind
}


// User does change the kind in a specialization.
template<> class vector<MyType> {
    static int iterator;
};

int main()
{
    printIter<MyType>();
}
```