

Allow structured bindings to accessible members

Timur Doumler (papers@timur.audio)

Document #: P0969R0
Date: 2018-03-14
Project: Programming Language C++
Audience: Evolution Working Group, Core Working Group

Abstract

This paper proposes to allow structured binding declarations to bind to a non-public member of a class, or to a member of its non-public base, if this member is accessible in the scope of the declaration. This makes structured bindings more consistent with other kinds of declarations.

1 The problem

In C++17, [del.struct.bind] specifies that a structured binding declaration can only bind to *public* data members of a class or one of its *public* base classes.

This rule introduces a weird inconsistency. For all other kinds of declarations, the ability to access a data member depends on the actual *accessibility* of that member in the current scope, and not on whether or not it was declared as public. This concept of accessibility is consistent throughout the language, except for structured bindings.

One case where this inconsistency manifests itself is accessing data members of a class from one of its friends, which grants access to non-public members — except for structured bindings:

```
struct A {
    friend void foo();
private:
    int i;
};

void foo() {
    A a;
    auto x = a.i; // OK
    auto [y] = a; // Ill-formed
}
```

Note that both declarations, `x` and `y`, try to access the same data member of the same object from the same scope. In one case it is allowed; in the other, it is not.

This problem also exists with non-public, but accessible base classes:

```
struct A {
    int i;
};
```

```

struct B : private A {
    friend void foo();
};

void foo() {
    B b;
    auto x = b.i; // OK
    auto [y] = b; // Ill-formed
}

```

In another variation of the same problem, it turns out that a class is not even allowed to bind to *its own* data members:

```

class C {
    int i;
    void foo(const C& other) {
        auto [x] = other; // Ill-formed
    }
};

```

This behaviour is confusing and inconsistent, and should be fixed. There are no good reasons for restricting access to members for structured binding declarations beyond the access rules that already exist for all other kinds of declarations in the language.

2 The solution

To make the above code well-formed, only a minimal change is needed. We need to remove the constraint that structured bindings can only bind to *public* data members of the class or its *public* base class. Instead, we only need the less strict requirement that those members shall be accessible in the context of the declaration, or in other words, that naming that member in the context of the declaration shall be well-formed. This will make the rules for accessing a class member name from a structured binding declaration consistent with accessing that name from any other kind of declaration.

3 Impact on the standard

The proposed change has no effect on C++ code that is already well-formed, since data members and base classes that are public are always accessible. It is purely an addition to allow code that would currently be ill-formed, i.e. data members and base classes that are accessible, but not public.

4 Proposed wording

The reported issue is intended as a defect report with the proposed resolution as follows. The effect of the wording changes should be applied in implementations of all previous versions of C++ where they apply. The changes are relative to the C++ working paper [Smith2018]. In [dcl.struct.bind], modify paragraph 4 as follows:

Otherwise, all of E’s non-static data members shall be ~~public~~-direct members of E or of the same ~~unambiguous-public~~-base class of E, well-formed when named as e.name in the context of the structured binding; E shall not have an anonymous union member~~;~~; and the number of elements in the *identifier-list* shall be equal to the number of non-static data members of E.

Acknowledgements

Many thanks to Alexander Kirillin and Herb Sutter for their very helpful comments and suggestions.

References

[Smith2018] Richard Smith. Working Draft, Standard for Programming Language C++. <https://github.com/cplusplus/draft>, 2018 (accessed 2018-03-14).