```
#include <utility>  // std::move

decltype(auto) v11;                                 // Error, no initializer
decltype(auto) v12 = f1(), v13 = std::move(i1);  // OK, deduced as int&&
decltype(auto) v14 = 5, v15 = f1();                 // Error, ambiguous deduction
```

A nonstatic data member cannot be declared using **decltype(auto)**, even if provided with a **default member initializer** (see Section 2.1."Default Member Init" on page 318):

```
struct C1
{
    decltype(auto) d_data = f();  // Error, decltype(auto) for data member
};
```

A **constexpr** static data member (see Section 2.1."**constexpr** Variables" on page 302) that is initialized at the point of declaration can be declared using **decltype(auto)**, but a non**constexpr** static data member cannot, simply because non**constexpr** static **members** must be initialized outside the class definition, independently of the **decltype(auto)** feature:

```
constexpr int f2() { return 5; }

struct C2
{
    static constexpr decltype(auto) s_mem1 = f2();  // OK
    static           decltype(auto) s_mem2 = f2();  // Error, in-class init
};
```

A variable with **static storage duration** (either in namespace or class scope) can be declared using an explicit type and then redeclared and initialized using **decltype(auto)**. Note, however, that some popular compilers reject these redeclarations[1]:

```
extern int gi;  // forward declaration

struct C3
{
    static decltype(f2()) s_mem1;  // type int
};

decltype(auto) gi =         f2();  // OK, compatible redeclaration
decltype(auto) C3::s_mem1 = f2();  // OK, compatible redeclaration
```

[1]GCC 10.2 (c. 2020) and MSVC 19.29 (c. 2021), among many other compilers, reject **auto** redeclaration of previously declared variables; see GCC bug 60352 (**pluzhnikov14**). However, nothing in the C++14 Standard appears to disallow such redeclarations, and an example in the C++20 Standard suggests that they are valid.