Nonetheless, even with this retroactive fix, the effects of the deduction rules when applied to braced-initializer lists might be puzzling. In particular, `std::initializer_list` is deduced when **copy initialization** is used instead of **direct initialization**, which requires including `<initializer_list>`:

```
auto x1 = 1;                 // int
auto x2(1);                  //  "
auto x3{1};                  //  "

#include <initializer_list>  // std::initializer_list
auto x4 = {1};               // OK, deduced as std::initializer_list<int>

auto x5{1, 2};               // Error, direct-list-init requires exactly 1 element.
auto x6 = {1, 2};            // OK, deduced as std::initializer_list<int>
```

Deducing built-in arrays is problematic

Deducing built-in array types using **`auto`** presents multiple challenges. First, declaring an array of **`auto`** is ill formed:

```
auto arr1[]  = {1, 2};  // Error, array of auto is not allowed.
auto arr2[2] = {1, 2};  // Error, array of auto is not allowed.
```

Second, if the array bound is not specified, either the program does not compile or `std::initializer_list` is deduced instead of a built-in array:

```
#include <initializer_list>  // std::initializer_list
auto arr3 = {1, 2};  // OK, deduced as std::initializer_list<int>
auto arr4{1, 2};     // Error, direct-list init requires exactly 1 element.
```

Finally, attempting to circumvent this deficiency by using an alias template (see Section 1.1. “**`using`** Aliases” on page 133) will result in code that compiles but has undefined behavior:

```
template <typename TYPE, std::size_t SIZE>
using BuiltInArray = TYPE[SIZE];

auto arr5 = BuiltInArray<int, 2>{1, 2};
    // Bug, taking the address of a temporary array
```

The type deduced for `arr5` is **`int`**`*` because array-to-pointer conversion is performed prior to deduction for nonreference types. ~~Binding a pointer to a temporary array~~ does not extend ~~its~~ lifetime, and the array is destroyed at the end of the full expression. Thus, any attempt to access elements of `arr5` will lead to undefined behavior. Furthermore, even if this trick were to work, such code would also almost entirely defeat the purpose of **`auto`** since neither the array element’s type nor the array’s bound would be deduced.