Just as how function template argument deduction never deduces a reference type for its by-value argument, a variable declared with an unqualified **`auto`** is never deduced to have a reference type:

```
int  val = 3;
int& ref = val;
auto tmp = ref;  // Type of tmp is deduced to be int, not int&.
```

Augmenting **`auto`** with reference qualifiers and cv-qualifiers, however, enables us to control whether the deduced type is a reference and whether it is **`const`** and/or **`volatile`**:

```
auto val = 3;
    // Type of val is deduced to be int,
    // the same as the argument for template <typename T> void deducer(T).

const auto cval = val;
    // Type of cval is deduced to be const int,
    // the same as the argument for template <typename T> void deducer(const T).

auto& ref = val;
    // Type of ref is deduced to be int&,
    // the same as the argument for template <typename T> void deducer(T&).

auto& cref1 = cval;
    // Type of cref1 is deduced to be const int&,
    // the same as the argument for template <typename T> void deducer(T&).

const auto& cref2 = val;
    // Type of cref2 is deduced to be const int&,
    // the same as the argument for template <typename T> void deducer(const T&).
```

Note that qualifying **`auto`** with `&&` does ~~*not*~~ result in deduction of an *rvalue* reference (see Section 2.1."*Rvalue* References" on page 710), but, in line with function template argument deduction rules, would be treated as a *forwarding reference* (see Section 2.1."Forwarding References" on page 377). A variable declared with **`auto&&`** will, therefore, result in an *lvalue* reference or an *rvalue* reference depending on the value category of its initializer:

```
double doStuff();

      int val  = 3;
const int cval = 7;

// Deduction rules are the same as for template <typename T> void deducer(T&&):

auto&& lref1 = val;
    // Type of lref1 is deduced to be int&.

auto&& lref2 = cval;
    // Type of lref2 is deduced to be const int&.
```