auto ignores top level cv-qualifiers

When auto sets the type of a declared variable from its initializing expression, it proceeds as follows:

1. If the initializing expression is a reference, the reference is ignored.
2. If, after Step 1 has been performed, there is a top-level const and/or volatilequalifier, it is ignored.

int& f ();

auto x = f (); // x is of type int, not int&

Think about the type of the local variable (x in our example) as having two parts: the core type (int in our case) and its const-ness/reference-ness. The core type can be naturally determined from the initializer expression. However, const/reference-ness is really determined by what we plan to do with the object further down within our code. Are we just accessing it? Then our variable should probably be a const reference. Are we planning to modify it? If so, then do we want to modify a shared object or our own copy? If it is shared, then our variable should be a reference. Otherwise, it should be a value. Here are the signatures for each case:

const auto& x = f (); // x is not modified

auto& x = f ();       // x is modified, shared

auto x = f ();        // x is modified, private

In a sense, by choosing to strip the top-level reference, auto forces us to specify our intentions. Plus, if we use the above signatures for each use-case, we get an additional safety net in case the type of an initializer changes.

Auto vs Decltype: Differences

int f1 ();

int& f2 ();

const int& f3 ();

auto a1 = f1 (); // a1 is int

auto a2 = f2 (); // a1 is int

auto a3 = f3 (); // a1 is int

decltype (f1 ()) d1 = f1 (); // d1 is int

decltype (f2 ()) d2 = f2 (); // d2 is int&

decltype (f3 ()) d3 = f3 (); // d3 is const int&