More generally, note that the && syntax can *never* imply a forwarding reference for a function that is not itself a template; see *Annoyances — Forwarding references look just like rvalue references* on page 397.

auto&& — a forwarding reference in a nonparameter context

Outside of template function parameters, forwarding references can also appear in the context of variable definitions using the **auto** keyword (see Section 2.1."**auto** Variables" on page 195) because they too are subject to type deduction:

```
void f()
{
    auto&& i = 0;  // i is a forwarding reference because the type of i must
                   // be deduced from the initialization expression 0.
}
```

Just like function parameters, **auto**&& resolves to either an lvalue reference or rvalue reference depending on the value category of the initialization expression:

```
void g()
{
    int i = 0;
    auto&& lv = i;  // lv is an int&.

    auto&& rv = 0;  // rv is an int&&.
}
```

Similarly to **const auto**&, the **auto**&& syntax binds to anything. In the case of **auto**&&, however, the reference will be **const** *only* if it is initialized with a **const** object:

```
void h()
{
    int        i = 0;
    const int ci = 0;

    auto&& lv  = i;   // lv is an int&.
    auto&& clv = ci;  // clv is a const int&.
}
```

Just as with function parameters, the original value category of the expression used to initialize a *forwarding* reference variable can be propagated during subsequent function invocation, e.g., using `std::forward` (see *The `std::forward` utility* on page 385):