```
decltype(lvref()) v1  = lvref();  // deduced as int&
decltype(auto)    v2  = lvref();  // equivalent to v1

decltype(rvref()) v3  = rvref();  // deduced as C1&&
decltype(auto)    v4  = rvref();  // equivalent to v3

decltype(c1)      v5  = c1;       // deduced as C1
decltype(auto)    v6  = c1;       // equivalent to v4

decltype((c1))    v7  = c1;       // deduced as C1&
decltype(auto)    v8  = (c1);     // equivalent to v7

decltype(cc1)     v9  = cc1;      // deduced as const C1
decltype(auto)    v10 = cc1;      // equivalent to v9
decltype((cc1))   v11 = cc1;      // deduced as const C1&
decltype(auto)    v12 = (cc1);    // equivalent to v11

decltype({ 3 })   v13 = { 3 };    // Error, not an expression
decltype(auto)    v14 = { 3 };    // Error, not an expression
```

The semantics of the **decltype** operator, when applied to an expression consisting of a single variable, cause **decltype**(c1) to yield type C1 and **decltype**((c1)) to yield reference type C1&, as in the definitions of v5 and v7, respectively; variables v6 and v8, therefore, also have the types C1 and C1&. A braced-initializer list such as { 3 } is not an expression; thus, v13 and v14 are both invalid.

Note that functions returning **scalar types** discard top-level cv-qualifiers on their return types, so a type deduced from a call to such a function will not reflect top-level cv-qualifiers even when defined with **decltype**(**auto**):

```
template <typename T> T f();

decltype(auto) v15 = f<const C1>();           // deduced as const C1
decltype(auto) v16 = f<const int>();          //    "     " int
decltype(auto) v17 = f<const int&>();         //    "     " const int&
decltype(auto) v18 = f<const char* const>();  //    "     " const char*
```

The top-level **const** qualifier on the class type, **const** C1, and on the reference type, **const int&**, are preserved but not on the scalar type, **const int**. The **const**ness of the pointer itself, in **const char*** **const**, is similarly discarded, as it is the top-level cv-qualifier on a scalar type.

When a function name is used as the initializer expression, it automatically decays to a pointer type when initializing a variable declared with type **auto** but does not decay when