```
Ili jL2 = ({1, 2, 3});  // Error, illegal context for statement expression
Ili jL2ne ({1, 2, 3});  // Bug, direct initialization from a copy
Ili jL3 = ((1, 2, 3));  // Error, conversion from int to nonscalar requested
Ili jL3ne ((1, 2, 3));  // Error, no matching function call for (int)

Ili kL4 = {{1, 2, 3}};  // Error, conversion from brace-enclosed list requested
Ili kL4ne {{1, 2, 3}};  // Error,      "       "    "       "       "      "
Ili kL5 = {(1, 2, 3)};  // Bug, copy initialization to single-int init list
Ili kL5ne {(1, 2, 3)};  // Bug, direct      "        "    "     "    "    "
```

As can be inferred from the code example above, the language treats direct and copy initialization of an `std::initializer_list` the same — i.e., as if the inaccessible constructor used by the compiler to populate an `std::initializer_list` is declared without the explicit keyword; see Section 2.1.“Braced Init” on page 215. If the list of values is enclosed in parentheses instead of braces, the list will be interpreted as either the use of the comma operator (`iL1`, `jL3`, `jL3ne`, `kL5`, and `kL5ne` above) or a function call (`iL1ne` above). Furthermore, it is important to avoid creating unnecessary copies, such as for `jL2ne` ~~above: If the copy is not elided by the compiler,~~ ~~`jL2ne`~~ refers to an array whose lifetime has ended.[2]

Annoyances

Initializer lists must sometimes be homogeneous

Though an `std::initializer_list<E>` is clearly always homogeneous, the initializer list used to create it in many cases can be a heterogeneous list of initializers convertible to the common type `E`. When the value type `E` needs to be deduced, however, the braced list must strictly be homogeneous:

```
#include <initializer_list>  // std::initializer_list

void f(std::initializer_list<int>) {}

template <typename E>
void g(std::initializer_list<E>) {}

int main()
{
    f({1, '2', 3});  // OK, heterogeneous list converts
    g({1, '2', 3});  // Error, cannot deduce heterogeneous list
    g({1,  2 , 3});  // OK, homogeneous list

    auto x = {1, '2', 3};  // Error, cannot deduce heterogeneous list
    auto y = {1,  2 , 3};  // OK, homogeneous list
    std::initializer_list<int> z = {1, '2', 3};  // OK, converts
}
```

[2] ~~In C++17, direct initialization of `std::initializer_list` from an implicitly created temporary `std::initializer_list` will always work due to guaranteed copy elision.~~