Ghost: Lvalues that do not designate an object (Updates N3701)

Document: N3740
Author: Martin Uecker
Date: 2025-12-28

Changes since N3701:

- Wording changes based on reflector discussion (keep “potentially”)
- Add a clarifying footnote
- Update to newest working draft (N3685)
- New discussion related to “Array subscription without decay” N3517

We have the following undefined behavior (J.2 item 18 in C23).

> 6.3.3.1 Lvalues, arrays, and function designators
>
> 1 An lvalue is an expression (with an object type other than void) that **potentially** designates an object; 55) **if an lvalue does not designate an object when it is evaluated, the behavior is undefined.**

But this appears to be a ghost, as there does not seem to be any possible way to evaluate an lvalue that does not designate an object without either violating a constraint or invoking undefined behavior already specified elsewhere. To see this, we consider all expressions that can be lvalues. Other types of expressions can not be lvalues. In particular, cast, comma, conditional, assignment expressions are not lvalues.

1. Identifier

```
foo
```

There needs to be a visible declaration when an identifier is used in an expression (6.5.2 Primary expressions, §2). For non-external definitions there are no forward declarations possible, so the declaration is always also the definition for the object. If an external declaration where there is no definition, then this is either a constraint violation (6.9 External definitions, 6.9.1 General, §3) or undefined behavior (6.9 External definitions, 6.9.1 General, §5).

2. String literal

```
“foo”
```

A string literal is an lvalue, but it is obvious that the object must exist.

3. Generic

```
_Generic(x, int *: *x)
```

The expression is an lvalue if the result expression is an lvalue (6.5.2.1 Generic selection, §4), so assuming that other types of lvalue must always designate an object, this is then also true for generic expressions.

4. Pointer

```
(*p)
```

If the pointer p does not point to an object, then there is undefined behavior (6.5.4.3 Address and indirection operators, §4).

5. Compound literal

```
(int[3]){ 0, 1, 2 }
```

A compound literal is an lvalue (6.5.3.6 Compound literals, §8), but here it is obvious that the object exists.

6. Structure of union member (.)

```
s.x
```

The expression is an lvalue when the first operand is an lvalue (6.5.3.4 Structure and union members, §3). Hence, the first operand designates an object when evaluated, and so does the result.

7. Structure of union member (→)

```
p→x
```

The phrasing “The value is that of the named member of the object to which the first expression points, and is an lvalue.” implies that there must a structure or union object the pointer points to. If no such object exists, there must be implicit undefined behavior.

8. Array subscripting

```
p[3]
```

This is the topic of DR166 and N823, but previously array subscription was defined to be equivalent pointer arithment, i.e. to *(p + 3) in this example which is undefined behavior if 3 is out-of-bounds. With N3517 we now have first-class array subscription if p is an array and not a pointer. For this case, UB for out-of-bounds subscription is also explicitly prescribed (6.5.3.2 Array subscripting, §4).

Suggested Change:

It is suggested to remove the vacuous undefined behavior in 6.3.3.1 and to make the undefined behavior in 6.5.3.4 explicit. Regarding the second change, one can argue that “shall point to an object” is still vague and that further conditions for undefined behavior as spelled out for the unary * operator are needed (which also needs improvements, see N3741). As this is unrelated to the main topic of this paper and similar wording is already used elsewhere in the standard, this is left for future improvement.

Wording Change 1 (relative to n3685)

6.3.3.1 Lvalues, arrays, and function designators

1 An lvalue is an expression (with an object type other than void) that potentially designates an object. [55] ~~**if an lvalue does not designate an object when it is evaluated, the behavior is undefined.**~~ **The expressions that are lvalues are specified throughout clause 6.5.**[XX]

XX) If an lvalue does not designate an object during lvalue conversion, either a constraint is violated or the behavior is undefined as specified throughout clause 6.5.

Annex J.2
~~**(15) An lvalue does not designate an object when evaluated (6.3.3.1).**~~

Wording Change 2 (relative to n3685)

6.5.3.4 Semantics

4 A postfix expression followed by the -> operator and an identifier designates a member of a structure or union object. **If the first expression does not point to a structure or union object, the behavior is undefined.** The value is that of the named member of the object to which the first expression points, and is an lvalue.95) If the first expression is a pointer to a qualified type, the result has the so-qualified version of the type of the designated member.

Annex J.2
(XX) In a postfix expression followed by the -> operator, the first expression does not point to a structure or union object.