Clarifying Undefined Behavior for the Indirection Operator

Document: N3711
Author: Martin Uecker
Date: 2025-09-07

The wording for the indirection operator is rather vague, taking about “invalid value” for a pointer. The problem with this is that the term “invalid” depends on context. Here, it is further clarified with a footnote, but unfortunately the footnote is also vague as it starts with “Among the invalid values..” and then lists only null pointers, pointers after the lifetime of the object has ended, and misaligned pointers, but there are two other cases. For the special case of an one-after pointer undefined behavior for the indirection operator is specified in the rules for pointer arithmetic, but it seems more logical to move it here. The wording also does not cover the situation where there is insufficient space, i.e. when the pointer points near the end of some allocation, but the object is larger than the remaining space. TS 6010 has improved wording but also misses this case.

The suggested wording (on the next page) lists all cases where the indirection operator has UB explicitly in the normative text. We also suggest to preserve the footnote and list all other scenarios where there was undefined behavior already before in the construction or use of the pointer (and which are therefor covered elsewhere).

Wording Change (relative to n3550)

6.5.4.3 Address and indirection operator

Semantics

…

4 The unary * operator denotes indirection. If the operand points to a function, the result is a function designator; if it points to an object, the result is an lvalue designating the object. If the operand has type "pointer to type", the result has type "type". ~~**If an invalid value value has been assigned to the pointer the behavior of the unary * operator is undefined.**~~ **The pointer shall not be a null pointer and shall not point to one past the last element of an array object. It shall be correctly aligned for "type" and shall point to a location in the address space with sufficient space for an object of this type.**102)

102) …
~~**Among the invalid values for dereferencing a pointer by the unary * operator are a null pointer, an address inappropriately aligned for the type of object pointed to, and**~~ **If** the address of an object after the end of its lifetime, **then any use of the pointer value in an evaluation including its use as an operand of the unary * operator that is evaluated has undefined behavior (see 6.2.4). Pointer arithmetic that results in a pointer that does not point to an element of the same array object or one past the last element of the array object has undefined behavior (see 6.5.7). Conversion of a pointer to a type for which is not correctly aligned has undefined behavior (see 6.3.3.3).**

6.5.7 Additive operators

Semantics

…

9 ... If the pointer operand is not null, and the pointer operand and result do not point to elements of the same array object or one past the last element of the array object, the behavior is undefined.105) If the addition or subtraction produces an overflow, the behavior is undefined. ~~**If the pointer operand is null or the result points one past the last element of the array object, it shall not be used as the operand of a unary * operator that is evaluated.**~~**XXX)**

XXX) If the the result points one past the last element of the array object, using it as an operand of the unary * operator that is evaluated has undefined behavior (see 6.5.4.3).

Annex J.2

(33) The operand of the unary * operator has ~~**an invalid value**~~ **is a null pointer or points to one past the last element of an array object, or is not correctly aligned, or points to a location in the address space with sufficient space.** (6.5.4.3)

~~**(42) Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that points just beyond the array object and is used as the operand of a unary * operator that is evaluated (6.5.7)**~~