Improved Type Safety for Null Pointer Constants

Martin Uecker, Graz University of Technology

Number: n3397
Date: 2024-11-23

C23 added *nullptr* and *nullptr_t* to improve type safety. The motivation was to improve type safety when null pointer constants were used. While these new language features do provide a possible path forward and C++ compatibility, they did not *per se* eliminate the type safety issue caused by allowing integer constant expressions of value zero to be used as null pointer constants. This leaves a type safety issue unaddressed which is often perceived as serious[1][2]

GCC has an '-Wzero-as-null-pointer-constant' that warns about the use of zero as a null pointer constant in C++ and this option will be available also C in GCC 15. The introduction of this option for C into the development branch was well received. The use with legacy code requires fixes similar to the removal of implicit int or similar type safety improvements in C, but such changes generally seem worthwhile to improve the quality of the code and preserve the long-term value of the code.

The author's preferred solution (Alternative 1) is to simply and directly remove the possibility to use integer constant expressions of value zero as null pointer constant from C2Y. An implementation would then also have to define *NULL* accordingly as *(void*)0*. Theoretically, we could also allow *NULL* to be defined as *nullptr*, but still allowing any discrepancy in the type of this constant between implementations would seem questionable. As the most common practice is to define this constant to *(void*)0* (as in POSIX) this seems to be the only reasonable choice that does not risk breaking too much existing code.

Alternative 1 would lead to (desired) constraint violations in code such as:

```
int *x = 0;
1 ? 0 : x;
struct { void *p; int a; } y = { 0 };
```

I should be noted that even when these come constraint violations, this only requires that conforming compilers issue a diagnostic. Consequently, compilers are still allowed to accept this code with a warning. Many compilers typically also have features to turn selected warnings off, only emit some warning in a 'pedantic' mode, or switch to older language mode. Thus, while the code would not be conforming to ISO C2Y anymore – which is expected to be published in a couple of years -, in practice, it would still be a soft change that would not directly break all C code that uses zero as a null pointer constant.

As a second option (Alternative 2), we provide wording which would make the use of integer constant expressions of value zero as a null pointer constant an obsolescent feature – hoping that compilers would start to warn about this.

---

1 Linus Torvalds: "Any language which allows you to write … is simply not worth using." https://www.spinics.net/lists/linux-sparse/msg10066.html

2 For a life demonstration where an experience C programmer gets bitten by this: https://www.youtube.com/watch?v=3D_h2RE0o0E&t=8368s

Proposed Wording, Alternative 1 (relative to n3301)

6.3.3.3p3

An integer constant expression with the value 0~~, such an expression~~ cast to type void *, or the predefined constant nullptr is called a null pointer constant.

7.21
4 The macros are
NULL
which expands to ~~an implementation-defined null pointer constant~~ **(void*)0**;

6.5.4.4 Unary arithmetic operators

Semantics

5 The result of the logical negation operator ! is 0 if the value of its operand ~~compares unequal to 0~~ **yields true when converted to bool**, 1 if the value of its operand ~~compares equal to 0~~ **yields false when converted to bool**. The result has type int. The expression !E is equivalent to ~~(0==E)~~ **(false == (bool)E)**

Further changes: In 6.5.14, 6.5.15, 6.5.16, 6.7.12, 6.8.5.2, 6.8.6.1 (including footnote 187), 7.2.2.1, the phrase “compares equal to 0” needs to be replaced with “yields false when converted to bool” and the phrase “compares unequal to 0” needs to be replaced with “yields true when converted to bool”.

Proposed Wording, Alternative 2 (relative to n3301)

7.21
4 The macros are
NULL
which expands to ~~an implementation-defined null pointer constant~~ **(void*)0**;

6.11.X Null Pointer Constants

The use of integer constant expressions of value 0 that are not cast to void* as null pointer constants is an obsolescent feature.