COMPARISON WITH LITERAL 0 ?!?

Martin Hořeňovský

PEX

COMPARISON WITH LITERAL 0 ?!?

Martin Hořeňovský

PEX

C++20 added the spaceship operator

C++20 added the spaceship operator

```
if ((a <=> b) == 0) {
  // ...
}
```

The result of <=> can only be compared with literal 0

The result of <=> can only be compared with literal 0

cmp.categories.pre p3

The result of <=> can only be compared with literal 0

cmp.categories.pre p3

The relational and equality operators for the comparison category types are specified with an anonymous parameter of unspecified type. This type shall be selected by the implementation such that these parameters can accept literal `0` as a corresponding argument.

The result of <=> can only be compared with literal 0

cmp.categories.pre p3

The relational and equality operators for the comparison category types are specified with an anonymous parameter of unspecified type. This type shall be selected by the implementation such that these parameters can accept literal `0` as a corresponding argument.

In this context, the behavior of a program that supplies an argument other than a literal `0` is undefined.

The details are left to the stdlib ...

The details are left to the stdlib ...

... so how does a library do this?

CONSTEXPR TRICK

```
struct ZeroLiteralAsPointer {

    constexpr
    ZeroLiteralAsPointer( ZeroLiteralAsPointer* ) noexcept {}

    template <typename T,
              typename = std::enable_if_t<
                  !std::is_same<T, int>::value
    >>
    constexpr ZeroLiteralAsPointer( T ) = delete;
};
```

CONSTEVAL TRICK

```
void ZeroLiteralErrorFunc();

struct ZeroLiteralConsteval {
    template <class T,
              std::enable_if_t<std::is_same_v<T, int>, int> = 0>
    consteval ZeroLiteralConsteval( T zero ) noexcept {
        if ( zero != 0 ) {
            ZeroLiteralErrorFunc();
        }
    }
};
```

COMPARISON

	`constexpr`	`consteval`
standard required	11	20 (23)
SFINAE-friendly	✓	✗
disallows `1-1`	✓	✗
Used by MS STL	⚠	✓
Used by libc++	⚠	✓
Used by libstdc++	✓	✗

⚠ - Used previously, not anymore

Why are stdlibs moving to `consteval`?

Why are stdlibs moving to `consteval`?

It seems worse in all aspects ...

Why are stdlibs moving to `consteval`?

It seems worse in all aspects ...

... except one.

Why are stdlibs moving to `consteval`?

It seems worse in all aspects ...

... except one.

A, very, very, very, dumb one.

C++ source #1

Save/Load + Add new... Vim CppInsights Quick-bench

```cpp
#include <compare>

struct ZeroLiteralAsPointer {

    constexpr
    ZeroLiteralAsPointer( ZeroLiteralAsPointer* ) noexcept {}

    template <typename T,
              typename = std::enable_if_t<
                  !std::is_same<T, int>::value
    >>
    constexpr ZeroLiteralAsPointer( T ) = delete;
};

int main() {
    ZeroLiteralAsPointer detector(0);
}
```

Output of x86-64 clang 19.1.0 (Compiler #1)

Wrap lines Select all

```
<source>:16:35: warning: zero as null pointer constant [-Wzero-as-null-pointer-constant]
   16 |     ZeroLiteralAsPointer detector(0);
      |                                   ^
      |                                   nullptr
1 warning generated.
Compiler returned: 0
```

I am not kidding: Microsoft/STL#3581

Fun fact: MSVC does not implement P2564

Fun fact: MSVC does not implement P2564

¯_(ツ)_/¯

CONCLUSION

We should stop standardizing magic library types

We should stop standardizing magic library types

Warnings can actively worsen everyone's code

Spaceship & comparison changes are a mess

Spaceship & comparison changes are a mess

See Catch2 for more fun issues

THE END