Formatting pointers

Document #: P2510R3
Date: 2022-05-15
Audience: LWEG
Reply-to: Mark de Wever
<koraq@xs4all.nl>

1 Revision History

Since [P2510R2]:

— Added 2022-05-03 LEWG polls.

— Removed proposed changes for the # formatting field (based on LWEG feedback).

Since [P2510R1]:

— Fixed formatting errors in tony tables.

— Provided a reference implementation.

Since [P2510R0] addressing LWEG weekly review feedback:

— Removed the proposed changes for the L formatting field.

— Added proposed changes for the # formatting field.

— A minor grammar fix to the proposed wording.

2 Polls

2.1 2022-05-03 LEWG (R2)

Poll: The # specifier should not be valid for pointers

SF	F	N	A	SA
2	8	1	3	0

Outcome: Weak consensus in favor

Poll: Modify [P2510R2] (Formatting pointers) by making the '#' specifier not valid for pointers, and then send the revised paper to LWG for C++26 classified as B2 (to be confirmed with a Library Evolution electronic poll).

SF	F	N	A	SA
2	13	1	0	0

Outcome: Strong consensus in favor

3 Introduction

The number of formatting options for pointer types is limited when compared to integer types. Since the formatting options are already implemented for integer types, some of these restrictions seem unnecessary and inconsistent. This paper aims to make formatting pointer types more useful, reducing the need for users to write their own formatters or casting a pointer type to an integer type.

4 Motivation and scope

Assuming `uintptr_t` is defined, a pointer is formatted like

```
to_chars(first, last, reinterpret_cast<uintptr_t>(value), 16)
```

with the prefix `0x` added to the output. When formatting an unsigned integer several *std-format-spec* fields can affect the output.

```
int i = 0;
format("{:#018x}", reinterpret_cast<uintptr_t>(&i)); // 0x00007ffe0325c4e4
format("{:#018X}", reinterpret_cast<uintptr_t>(&i)); // 0X00007FFE0325C4E4
```

The latter example isn't possible when using a pointer type directly. When the user wants this output they need to use a `reinterpret_cast` or write their own formatter.

Whether the first example is possible when using a pointer type directly depends on whether the hexadecimal output of the pointer type is considered an integer presentation type. [LWG3612] asserts it's not. Currently an integer presentation type isn't defined in the standard, [LWG3644] proposes to add this definition and excludes pointer types. That solves the ambiguity for the pointer type, but means all three examples aren't possible for pointer types.

In order to make the formatting of pointer types useful and more consistent all *std-format-spec* fields are evaluated.

fill
No changes are proposed.

align
[LWG3612] addresses the default alignment of pointer types. No changes are proposed.

sign
This won't be allowed after [LWG3644]. Since pointers aren't arithmetic types they shouldn't have a sign. No changes are proposed.

#
This won't be allowed after [LWG3644]. Allowing this has no effect since the output always has a base prefix. No changes are proposed.

0
This won't be allowed after [LWG3644], before its status was unclear. Zero-padding a pointer type to a requested width is useful and matches the behaviour of the integer presentation type. Therefore the field will be allowed for pointer types.

width
No changes are proposed.

precision
No changes are proposed.

L
No changes are proposed.

type
Currently it's only possible to generate lower case output. The other hexadecimal output types `x` and `a` have an upper case equivalent `X` and `A`. For consistency the pointer type should have an upper case equivalent. This paper proposes the type `P` for upper case output of the pointer type.

The wording of `p` uses "prefix", while the wording of `0` applies to a "base prefix". For clarity and consistency this will be adjusted so the wording in [tab:format.type.ptr] matches the wording in [tab:format.type.int].

The summary of the proposed changes where `ptr` is a pointer type:

Format	Before	After
`format("{:018}", ptr);`	Unclear, ill-formed after [LWG3644]	0x00007ffe0325c4e4
`format("{:P}", ptr);`	Ill-formed	0X7FFE0325C4E4
`format("{:-}", ptr);`	Unclear, ill-formed after [LWG3644]	Unclear, ill-formed after [LWG3644]
`format("{:#}", ptr);`	Unclear, ill-formed after [LWG3644]	Unclear, ill-formed after [LWG3644]

5 Impact on the Standard

The paper only proposes library extensions to the `<format>` header. The proposed changes are already in used for integer types. Based on the scope of the changes and my implementation experience with `<format>` in libc++ I foresee no issues implementing this paper. A libc++ based reference implementation is available.

6 Proposed wording

Based on [N4901]:

Update the value of the feature-testing macro `__cpp_lib_format` to the date of adoption in [version.syn].

20.20.2.2 Standard format specifiers [format.string.std]

1 ...

type: one of
`a A b B c d e E f F g G o p P s x X`

13 A zero (`0`) character preceding the *width* field pads the field with leading zeros (following any indication of sign or base) to the field width, except when applied to an infinity or NaN. This option is only valid for arithmetic types other than `charT` and `bool`, pointer types, or when an integer presentation type is specified. If the `0` character and an *align* option both appear, the `0` character is ignored.

[Editor's note: The wording modification is the same as used in [LWG3612].]

Table 69: Meaning of *type* options for pointer types [tab:format.type.ptr]

Type	Meaning
none, `p`	If `uintptr_t` is defined, `to_chars(first, last, reinterpret_cast<uintptr_t>(value), 16)` with the base prefix `0x` added to the output; otherwise, implementation-defined.
`P`	The same as `p`, except that it uses uppercase letters for digits above 9 and the base prefix is `0X`.

Refences

[LWG3612] Victor Zverovich. *Inconsistent pointer alignment in std::format*. Oct. 2, 2021. URL: https://wg21.link/LWG3612.

[LWG3644] Charlie Barto. *std::format does not define "integer presentation type"*. Nov. 23, 2021. URL: https://wg21.link/LWG3644.

[N4901] Thomas Köppe. *Working Draft, Standard for Programming Language C++*. Oct. 23, 2021. URL: https://wg21.link/N4901.

[P2510R0] Mark de Wever. *Formatting pointers*. Dec. 13, 2021. URL: https://wg21.link/p2510r0.

[P2510R1] Mark de Wever. *Formatting pointers*. Mar. 13, 2022. URL: https://wg21.link/p2510r1.

[P2510R2] Mark de Wever. *Formatting pointers*. Apr. 13, 2022. URL: https://wg21.link/p2510r2.