Formatting pointers

Document #:	P2510R0
Date:	2021-12-13
Audience:	LWG

Reply-to:	Mark de Wever <koraq@xs4all.nl>

1 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.

2 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
format("{:#Lx}", reinterpret_cast<uintptr_t>(&i));   // 0x7ffe_0325_c4e4
```

The latter two examples aren'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
This is currently not allowed. It gives the user the option to use "thousand" separators increasing the readability of the output. Allowing this is useful and matches the behaviour of the integer presentation type. Therefore the field will be allowed for pointer types.

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(":L", ptr);`	Ill-formed	0x7ffe_0325_c4e4
`format(":-", ptr);`	Unclear, ill-formed after [LWG3644]	Unclear, ill-formed after [LWG3644]
`format(":#", ptr);`	Unclear, ill-formed after [LWG3644]	Unclear, ill-formed after [LWG3644]

3 Impact on the Standard

The paper only proposes library extensions to the `<format>` header. There's currently no implementation experience. 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.

4 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].]

15 When the `L` option is used, the form used for the conversion is called the *locale-specific form*. The `L` option is only valid for arithmetic or pointer types, and its effect depends upon the type.

(15.1) — For integral or pointer types, the locale-specific form causes the context's locale to be used to insert the appropriate digit group separator characters.

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.