C2x Issue Report
WG14 N2420

Title:	Unclear type relationship between a format specifier and its argument
Author, affiliation:	Aaron Ballman, GrammaTech
Date:	2019-09-05
Proposal category:	Change/Clarification Requests

Abstract: The type relationship between an argument passed to a formatted io function and its corresponding format specifier could stand to be clarified.

Unclear type relationship between a format specifier and its argument

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2420
Date: 2019-09-05

Summary of Changes

N2420

- Original report

Introduction and Rationale

7.21.6.1p9 states:

> 9 If a conversion specification is invalid, the behavior is undefined.[286)] If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.

The second sentence does not make it clear what constitutes a "correct type". For instance, given that there is no conversion specification for objects of type `_Bool`, this sentence could be read that passing an object of type `_Bool` to `printf()` is undefined behavior.

Reflector discussion (from roughly SC22WG14.17025 through SC22WG14.17060) observed that we have concepts like exact type matches, same representation, and compatible types, and that it would be better to reword this paragraph to bring it in line with `<stdarg.h>` instead of doing a minor correction. The ensuing reflector discussion found additional confusion in that the `%n` specifier does not explicitly state its type requirements in the absence of a length modifier, nor does `%p` suggest that it can be used to print a `const`-qualified pointer to non-`void` type, such as `const char *`.

Proposed Wording

The wording proposed is a diff from ISO/IEC 9899-2017. Green text is new text, while ~~red~~ text is deleted text.

Modify 7.21.6.1p8, the `p` and `n` specifiers:

8 The conversion specifiers and their meanings are:

…

`p` The argument shall be a pointer to `void` or a pointer to a character type. The value of the pointer is converted to a sequence of printing characters, in an implementation-defined manner.

`n` The argument shall be a pointer to signed integer whose type is specified by the length modifiers, if any, for the conversion specification, or shall be `int` if no length modifiers are specified for the conversion specification. ~~into which is written t~~The number of characters written to the output stream so far by this call to `fprintf` will be stored into the integer pointed to by the argument. No argument is converted, but one is consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.

…

Modify 7.21.6.1p9:

9 If a conversion specification is invalid, the behavior is undefined.[286)] ~~If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.~~

Append to that same paragraph:

For an integer argument, the promoted type of the argument shall have the same integer rank as the promoted type corresponding to the conversion specification, and the value to be printed is determined as if converting to the unpromoted type corresponding to the conversion specification.[x)] An argument that corresponds to a conversion specification that requires a pointer to `void` (`p`) or a pointer to character type (`s` without the `l` modifier) shall have a type that is a pointer to a qualified or unqualified version of `void` or of a character type. Otherwise, after default argument promotions, the type of the argument shall be the same as the type corresponding to the conversion specification including possible length modifiers.

Add a new footnote:

[x)] The behavior is undefined if the value does not fit the type corresponding to a conversion specification with a signed integer type.

Modify 7.21.6.2p12:

12 In the following, the type of the corresponding argument for a conversion specifier shall be a pointer to a type determined by the length modifiers, if any, or shall be a pointer to `int`, `unsigned int`, or `float` for signed integer, unsigned integer and floating conversion, respectively. The conversion specifiers and their meanings are:

…

`n` No input is consumed. The corresponding argument shall be a pointer to signed integer .~~into which is to be written t~~The number of characters read from the input stream so far by this call to the `fscanf` function will be stored into the integer pointed to by the argument. Execution of a `%n` directive does not increment the assignment count returned at the completion of execution of the `fscanf` function. No argument is converted, but one is consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.

Acknowledgements

I would like to recognize the following people for their help with this work: Jens Gustedt, Martin Uecker, Joseph Myers, and Martin Sebor.