C2x Issue Report
WG14 N2562

Title: Unclear type relationship between a format specifier and its argument
Author: Aaron Ballman
Date: 2020-09-01
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: N2562
Revises Document No: N2483
Date: 2020-09-01

Summary of Changes

N2562

- Minor editorial cleanups
- Changed “stored into the integer pointed to” into “stored into the integer object pointed to”
- Reformulated `va_arg` in the presence of no actual next argument.

N2483

- Clarified `va_arg` behavior for pointer qualifications
- Updated the wording to be based on `<stdarg.h>` and `va_arg`
- Updated the specifications for `fscanf`, `fwprintf` and `fwscanf`

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 WG14 N2454. Green text is new text, while ~~red~~ text is deleted text.

Modify 7.16.1.1p2:

The `va_arg` macro expands to an expression that has the specified type and the value of the next argument in the call. The parameter `ap` shall have been initialized by the `va_start` or `va_copy` macro

(without an intervening invocation of the `va_end` macro for the same `ap`). Each invocation of the `va_arg` macro modifies `ap` so that the values of successive arguments are returned in turn. The behavior is undefined if there is no actual next argument. The parameter *type* shall be a type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a `*` to *type*. ~~If there is no actual next argument, or if~~ If *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined, except for the following cases:

— both types are pointers to qualified or unqualified versions of compatible types;

— one type is a signed integer type, the other type is the corresponding unsigned integer type, and the value is representable in both types;

— one type is pointer to qualified or unqualified `void` and the other is a pointer to a qualified or unqualified character type.

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` is stored into the integer object 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:

`fprintf` shall behave as if it uses `va_arg` with a type argument naming the type resulting from applying the default argument promotions to the type corresponding to the conversion specification and then converting the result of the `va_arg` expansion to the type corresponding to the conversion specification.[x)]

Add a new footnote:

[x)] The behavior is undefined when the types differ as specified for `va_arg` (7.16.1.1).

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 specified by the conversion specifier. The conversion specifiers and their meanings are:

`d` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `int` ~~signed integer~~.

`i` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `int` ~~signed integer~~.

`o` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `unsigned int` ~~unsigned integer~~.

`u` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `unsigned int` ~~unsigned integer~~.

`x` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `unsigned int` ~~unsigned integer~~.

`a,e,f,g` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `float` ~~floating~~.

`c` …

If no `l` length modifier is present, the corresponding argument shall be a pointer to `char`, `signed char`, `unsigned char`, or `void` that points to a buffer ~~the initial element of a character array~~ large enough to accept the sequence. No null character is added.

…

`s` …

If no `l` length modifier is present, the corresponding argument shall be a pointer to `char`, `signed char`, `unsigned char`, or `void` that points to a buffer ~~the initial element of a character array~~ large enough to accept the sequence and a terminating null character, which will be added automatically.

…

`[` …

If no `l` length modifier is present, the corresponding argument shall be a pointer to `char`, `signed char`, `unsigned char`, or `void` that points to a buffer ~~the initial element of a character array~~ large enough to accept the sequence and a terminating null character, which will be added automatically.

…

…

`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 is stored into the integer object 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.

Modify 7.29.2.1p8, the p and n specifiers:

…

`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 wide 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 wide characters written to the output stream so far by this call to `fwprintf` is stored into the object 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 7.29.2.1p9:

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:

`fwprintf` shall behave as if it uses `va_arg` with a type argument naming the type resulting from applying the default argument promotions to the type corresponding to the conversion specification and then converting the result of the `va_arg` expansion to the type corresponding to the conversion specification.[x)]

Add a new footnote:

[x)] The behavior is undefined when the types differ as specified for `va_arg` (7.16.1.1).

Modify 7.29.2.2p12:

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 specified by the conversion specifier. The conversion specifiers and their meanings are:

`d` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `int` ~~signed integer~~.

`i` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `int` ~~signed integer~~.

`o` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `unsigned int` ~~unsigned integer~~.

`u` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `unsigned int` ~~unsigned integer~~.

`x` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `unsigned int` ~~unsigned integer~~.

`a,e,f,g` … Unless a length modifier is specified, t~~T~~he corresponding argument shall be a pointer to `float` ~~floating~~.

`c` …

If no `l` length modifier is present, … The corresponding argument shall be a pointer to `char`, `signed char`, `unsigned char`, or `void` that points to a buffer ~~the initial element of a character array~~ large enough to accept the sequence. No null character is added.

…

`s` …

If no `l` length modifier is present, … The corresponding argument shall be a pointer to `char`, `signed char`, `unsigned char`, or `void` that points to a buffer ~~the initial element of a character array~~ large enough to accept the sequence and a terminating null character, which will be added automatically.

…

`[` …

If no `l` length modifier is present, … The corresponding argument shall be a pointer to `char`, `signed char`, `unsigned char`, or `void` that points to a buffer ~~the initial element of a character array~~ large enough to accept the sequence and a terminating null character, which will be added automatically.

…

…

`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 wide characters read from the input stream so far by this call to the `fwscanf` 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 `fwscanf` function. No argument is converted, but one is consumed. If the conversion specification includes an assignment-suppressing wide 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, Robert Seacord, and Martin Sebor.