C23 proposal: formatted input/output of binary integer numbers

WG14 N2612

Title:	Formatted input/output of binary integer numbers
Author:	Jörg Wunsch `<j.gnu@uriah.heep.sax.de>`
Date:	2020-11-10
Proposal category:	New feature
Reference:	N2473 C2x Working Draft

Abstract:

As a logical counterpart to allowing for binary constants in source code (N2549), it seems a logical consequence to also provide a method for formatted input/output of binary integer numbers.

While there is not much prior art known in this respect for C (or C++) implementations, other languages (Python, Rust, Perl) already offer this feature.

Background

When adopting N2549 (binary integer constants in C source code), a request arose to also consider the ability to allow for formatted input/output of binary integer numbers in the C language. So, in particular, `printf` should be able to e.g. output `101010` for an integer constant 42, and `scanf` is supposed to be able to parse `0b11` as number 3.

State of the Art

There are no major C (or C++) implementations known (as the time of this writing) that implement these features.

However, other languages implement it. There appears to be an agreement to use the format specifier `b` for it.

```
$ python -c 'print("{:b}".format(42))'
101010
$ perl -e 'printf("%b\n", 42);'
101010
$ cat main.rs
fn main() {
    println!("{:b}", 42);
}
$ rustc main.rs
$ ./main
101010
```

`printf`- and `scanf`-like functions

In addition to the standard number formatting, `printf` and relatives offer an *alternate* formatting option, designated by the modifier `#`. This causes the resulting string to be preceded by `0x` (for hexadecimal output), or `0b` (in the binary case).

If the value to be printed is 0, the hexadecimal formatting does not print the prefix though. Python and Rust handle that differently, and always print the prefix, which might seem more useful. For C, it is too late to change this behaviour as it would cause existing code to break – at least for hexadecimal output. For binary output, it could be handled differently, but that would make it inconsistent with the hexadecimal option. Thus, it is proposed to have the `#b` formatting behave similar to `#x`, and do not print the prefix.

Likewise, using an uppercase `#X` instead of `#x` causes the prefix (and, in the hexadecimal case, the digits `A` through `F`) to be printed in uppercase letters.

Ideally, the same would be done for an uppercase `#B` specifier for binary numbers. However, §7.31.13 only reserves lowercase letters for future library use; thus, an implementation could have been using uppercase `B` for their own extension already right now.

It is therefore proposed to suggest that an uppercase `B` format specifier can either be used for printing binary numbers, where the prefix in the alternate form becomes `0B`, or it can be handled in an implementation-defined manner. That way, any existing implementation already using it would not need to be changed. As a consequence, portable code could not rely on `#B` printing a `0B` prefix, but that seems to be tolerable. Portable code could always use a standard `b` specifier, and manually prepend `0B` if desired:

```
    printf("0B%08b\n", some_number);
```

For `scanf`-like functions, there is no need to add the uppercase `B` specifier. Prefixes like `0x` or `0B` are always allowed (for the respective formats), without distinguishing whether the format specifier is given in lower or upper case.

`strto*` functions

It is proposed to extend these functions in a similar way. That is, if the conversion base is 0, an initial `0b` or `0B` prefix causes the number to be interpreted as a binary format. If the conversion base is 2, the prefix is allowed but not necessary. For any base of 12 or above, an initial `0b` or `0B` string would be interpreted as digit `0`, followed by digit `B` (with value 11).

Suggested changes:

Additions are marked in green.

§7.21.6.1 The `fprintf` function

(4) third list item

An optional *precision* that gives the minimum number of digits to appear for the `b`, `B`, `d`, `i`, `o`, `u`, `x`, and `X` conversions, …

(6) item

The result is converted to an ”alternative form”. For `o` conversion, it increases the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both 0, a single 0 is printed). For `b` (or `B`) conversion, a nonzero result has `0b` (or `0B`) prefixed to it. For `x` (or `X`) conversion, a nonzero result has `0x` (or `0X`) prefixed to it. …

(6) item `0`

For `b`, `B`, `d`, `i`, `o`, `u`, `x`, `X`, `a`, `A`, `e`, `E`, `f`, `F`, `g`, and `G` conversions, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the `0` and `-` flags both appear, the `0` flag is ignored. For `b`, `B`, `d`, `i`, `o`, `u`, `x`, and `X` conversions, if a precision is specified, the `0` flag is ignored. For other conversions, the behavior is undefined.

(7) items `hh`, `h`, `l`, `ll`, `j`, `z`, `t`

Specifies that a following `b`, `B`, `d`, `i`, `o`, `u`, `x`, or `X` conversion …

(8) second item

`b`, `B`, `o`, `u`, `x`, `X`	The **unsigned int** argument is converted to unsigned binary (`b` or `B`), unsigned octal (`o`), unsigned decimal (`u`), or unsigned hexadecimal notation (`x` or `X`) in the style *dddd*; the letters `abcdef` are used for `x` conversion and the letters `ABCDEF` for `X` conversion. . . . Alternatively, conversion specifier `B` might be handled in an implementation-defined manner, differently from the description in the paragraphs above.

New (14) (in Recommended practice)

The option to handle a `B` conversion specifier in an implementation-defined manner allows for implementations that have implemented their own extension for it according to the rules of previous versions of this standard to continue using it the way they have been using it before. Implementations that did not historically use conversion specifier `B` should implement it as described in this standard.

§7.21.6.2 The `fscanf` function

(11) items `hh`, `h`, `l`, `ll`, `j`, `z`, `t`

Specifies that a following `b`, `d`, `i`, `o`, `u`, `x`, `X`, or `n` conversion . . .

(12) new first item

`b`	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the `strtoul` function with the value `2` for the `base` argument. The corresponding argument shall be a pointer to unsigned integer.

§7.22.1.7 The `strtol`, `strtoll`, `strtoul`, and `strtoull` functions

(3)

. . . The letters from `a` (or `A`) through `z` (or `Z`) are ascribed the values `10` through `35`; only letters and digits whose ascribed values are less than that of `base` are permitted. If the value of `base` is `2`, the characters `0b` or `0B` may optionally precede the sequence of letters and digits, following the sign if present. If the value of `base` is `16`, the characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.

§7.29.4.1.3 The `wcstol`, `wcstoll`, `wcstoul`, and `wcstoull` functions

(3)

. . . The letters from `a` (or `A`) through `z` (or `Z`) are ascribed the values `10` through `35`; only letters and digits whose ascribed values are less than that of `base` are permitted. If the value of `base` is `2`, the characters `0b` or `0B` may optionally precede the sequence of letters and digits, following the sign if present. If the value of `base` is `16`, the characters `0x` or `0X` may optionally precede the sequence of letters and digits, following the sign if present.