Document Number: N3183

Submitter: Aaron Peter Bachmann

Submission Date: 2023-11-24

Removing undesirable undefined behavior from `scanf()`.

Summary

It should be easy to use the functions of the standard-library to safely consume untrusted input. Presently for the scanf-family the behavior is undefined if the result of the conversion cannot be represented in the destination object. Since the input is not always under the complete control of the programmer the use of functions of the scanf-family of functions is problematic or cumbersome. This paper proposes to remove the undefined behavior if the input of numerical conversions does not fit into the destination. It also proposes to remove the undefined behavior for invalid wide character input.

Problem description for C23[1][2] and earlier

1. Character or text input:

There is no problem for character or text input provided the input characters fit into `char`, `signed char` or `unsigned char`. Using the optional maximum field width or the length modifier is sufficient to circumvent all problems.

2. Floating-point input - mostly no problem:

If ISO/IEC 60559 [4] floating-point numbers are used all input values are representable in the destination after rounding since rounding may round to +INF or -INF. NANs are representable as well. This also applies to many other floating-point representations. For less common floating-point formats this may be a (rare) problem. Only a portable method for distinguishing +INF or -INF form overflow is missing.

3. Integer input – a real problem:

J.2 Undefined behavior
(170)[1] The result of a conversion by one of the formatted input functions cannot be represented in the corresponding object, or the receiving object does not have an appropriate type (7.23.6.2, 7.31.2.2).

For integer input using the optional maximum field width or the length modifier can also effectively protect against converting values not representable in the target type. Unfortunately, this restricts the range of numbers convertible to a subset of the numbers representable in the target type. This is not particularly limiting for most applications, but more problematic for libraries, where the range of acceptable values may be unknown in advance.

For destination types smaller than `[unsigned] long long` it may be feasible to use the optional maximum field width or the length modifier, and then to convert to a smaller type after checking the conversion result represented as `[unsigned] long long` fits into the smaller type. This strategy

[1] This refers to ISO/IEC DIS 9899. It is (169) in n3096

is more likely to fail for `%...i` because the values representable in a certain number of characters varies very widely depending on the input base.

4. **Wide character Input – another real problem:**

J.2 Undefined behavior
(171) A c, s, or [conversion specifier with an l qualifier is encountered by one of the formatted input functions, but the input is not a valid multibyte character sequence that begins in the initial shift state (7.23.6.2, 7.31.2.2).

As the input may not be under the full control of the programmer – probably more often than not – there should be an innocuous way to detect this. Thus, `scanf()` - as specified by the C standard - is not well suited for multibyte character input.

Existing praxis

POSIX

`POSIX` [3] addresses the wide character problem. When the input does not form a valid character errno is set to `EILSEQ`. The POSIX specification makes sense, and it is existing practice. The C-Standard has `EILSEQ` in `errno.h` as well. Thus, C should follow `POSIX`. The fact that the C standard functions `mbrtowc()` has provisions to set `errno` to `EILSEQ` also suggests that this is the right path to follow.

Observed behavior of two C-libraries

The table below shows the actual observed behavior of `glibc (Debian GLIBC 2.36-8) 2.36` and `musl 1.2.3.-1`.

libc	scanf	input	i,d	r	errno
glibc	r=scanf("%d",&i);	2	2	1	0 (unmodified)
glibc	r=scanf("%d",&i);	99999999999999999999	-1	1	34 (ERANGE)
glibc	r=scanf("%d",&i);	-99999999999999999999	**0**	1	34 (ERANGE)
musl	r=scanf("%d",&i);	2	2	1	0 (unmodified)
musl	r=scanf("%d",&i);	99999999999999999999	-1	1	34 (ERANGE)
musl	r=scanf("%d",&i);	-99999999999999999999	**-1**	1	34 (ERANGE)
glibc	r=scanf("%le",&d);	1e4444	inf	1	34 (ERANGE)
glibc	r=scanf("%le",&d);	-1e4444	-inf	1	34 (ERANGE)
musl	r=scanf("%le",&d);	1e4444	inf	1	34 (ERANGE)
musl	r=scanf("%le",&d);	-1e4444	-inf	1	34 (ERANGE)

If two (or more) libraries do not agree there is no common existing practice. It seems both libraries use -1 as error indication. It looks like `glibc` negates after conversion (0 is the two's complement of -1)[2,3]. With the disagreement for signed integers, there is no need to look at the behavior for unsigned integers.

There are several sensible options for integer conversions here. The two most appropriate ones seem to be:

1. Leaving the result mostly unspecified on overflow. Only require it to be a value representation.

[2] This is a wild guess not actually verified by looking into the code. The implementation details do not matter for the paper.
[3] Negation after conversion is allowed. It is problematic for directed rounding modes for floating-point numbers. This is out of the scope of this document.

2. Requiring the result should approximate the actual value as good as possible, i. e. saturating at `INT_MIN, INTMAX, 0, UINTMAX, LONG_MIN, LONG_MAX, 0, ULONG_MAX, …`

Option 1 has fewer impact on existing implementations.

Option 2 provides more information for the user.

The author has a slight preference for Option 2.

`errno` should be set to `ERANGE` in any case of numerical overflow (integer and floating type).

Proposed solution:

1. Character or text input:

No action required.

2. Floating-point input:

There are no big problems with for most floating-point formats.

For floating point formats specified in ISO/IEC 60559 on overflow the value shall be set to `-inf` or `inf` with proper sign and `errno` shall be set to `ERANGE`.

For unusual and/or ancient floating-point representation this paper does not impose any requirements, but adds a recommended practice:

- The values are recommended to saturate at the highest or lowest value representable (including +-INF if they exist).
- In addition, it is recommended that `errno` shall be set to `ERANGE`, if the input value is not representable without saturation.

3. Integer input – the real problem:

Give an unspecified value, but not a non-value representation and indicate the overflow be setting `errno` to `ERANGE`.

4. Wide character input

Do it as POSIX does.

Wording changes:

The changes given here are relative to ISO/IEC DIS 9899:2023(E) for the scanf family:

In 7.23.6.2 after p9 insert a new paragraph:

If the input byte sequence for `lc`, `ls`, or `l[` does not form a valid multibyte character the conversion will fail, `errno` will be set to `EILSEQ` and the number of input bytes consumed will unspecified.

In 7.23.6.2 p10 replace

~~If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the object, the behavior is undefined.~~

with either

If this object does not have an appropriate type, the behavior is undefined.
If the result of the conversion cannot be represented in the object and the object has one of the floating-point formats specified in ISO/IEC 60559:2020 on overflow the value shall be set to -inf or inf with proper sign and `errno` shall be set to `ERANGE`.

If the result of the conversion cannot be represented in the object and the object has any integer type including bool, the object will have the representable value closest to the value representable[4] representation and `errno` shall ERANGE. Otherwise, the behavior is undefined.

or

If this object does not have an appropriate type, the behavior is undefined.
If the result of the conversion cannot be represented in the object and the object has one of the floating-pont formats specified in ISO/IEC 60559 on overflow the value shall be set to -inf or inf with proper sign and `errno` shall be set to `ERANGE`. If the result of the conversion cannot be represented in the object and the object has any integer type including bool, the object will be unspecified, but it will not be a non-value representation and `errno` shall `ERANGE`. Otherwise, the behavior is undefined.

and append

Recommended practice
For floating point formats not specified in ISO/IEC 60559:2020 on overflow the values are recommended to saturate at the highest or lowest value representable (including +-INF if they exist) and it is recommended that `errno` shall be set to `ERANGE`, if the input value is not representable without saturation.

Modify J.2
(170) The result of a conversion by one of the formatted input functions cannot be represented in the corresponding floating-point object for floating point types not covered by ISO/IEC 60559:2020, or the receiving object does not have an appropriate type (7.23.6.2, 7.31.2.2).
~~(171) A c, s, or [conversion specifier with an l qualifier is encountered by one of the formatted input functions, but the input is not a valid multibyte character sequence that begins in the initial shift state (7.23.6.2, 7.31.2.2).~~

Apply the same changes to `wscanf()`[5].

[4] `0, 1, CHAR_MIN, CHAR_MAX, UCHAR_MIN, UCHAR_MAX, SCHAR_MIN, SCHAR_MAX, …, INT_MIN, …`

[5] Since - as far as the author of this papers understands it, without having ever seen the source of the standard - it is generated from the same source no separate changes to the source of the standard are expected to be required for `wscanf()`.

Possible straw polls:

Depending on the discussion:

Decision poll:

1. Does the committee wish do adopt the changes proposed in N3183 with the preferences expressed in tiny round brackets in the following questions?
2. Do we want saturation for integer types? (yes)
3. Do we want unspecified values for integers on overflow? (yes)
4. Do we want saturation on overflow for floating point values? (yes)
5. Do we want `errno` set to `ERANGE` for integer types and ISO/IEC 60559 floating-point types and recommended practice for other floating-point types? (yes)
6. Do we want the POSIX behavior to avoid undefined behavior for invalid wide characters? (yes)

If the proposal is rejected, does the committee wish to see something along the lines of N3183?

Acknowledgements:

I want to thank Thomas Kemmer and Reinhard Kopka for helpful comments.

References:

[1] ISO/IEC DIS 9899 Information technology — Programming languages — C

[2] https://www.open-std.org/jtc1/sc22/wg14/www/docs/n3054.pdf

[3] P1003.1™-202x/D3 Draft Standard for Information Technology —
Portable Operating System Interface (POSIX™) ISO/IEC 9945-202x/D3, March 2023

[4] ISO/IEC 60559:2020 Information technology Microprocessor Systems Floating-Point arithmetic