generic_count_type, v1.0.0

WG14 N 3593

Title:

Author, affiliation: Robert C. Seacord,
Woven by Toyota,
rcseacord@gmail.com

Date: 2025-5-22

Proposal category: Defect

Target audience: Implementers, users

Abstract: This document proposes replacing the `generic_count_type` parameter in the `stdc_rotate_left` and `stdc_rotate_right` functions with an `unsigned int`. It argues that `unsigned int` is sufficient and avoids introducing unnecessary undefined behavior with negative counts.

Prior art: C2Y committee draft

generic_count_type, v1.0.0

Reply-to: Robert C. Seacord (rcseacord@gmail.com)

Document No: **N 3593**

Reference Document: **N 3467, P0586R2**

Date: 2025-3-20

This proposal changes the rule for forming composite types

Change Log

2025-3-23:

- Initial version 1.0.0

Table of Contents

1 Problem Description

N3367: More Modern Bit Utilities introduced the `stdc_rotate_left` and `stdc_rotate_right` functions along with the ***generic_count_type*** `count` parameter. This paper was accepted for C2Y, after C23 had already been published.

The ***generic_count_type*** `count` parameter is used to specify the number of positions the value should be left or right rotated. Notionally, it is an unsigned integer value. Conceivably, rotating a value left a negative number of bits could rotate the value right, and rotating a value right a negative number of bits could rotate the value left. This approach might make more sense if there was only one `stdc_rotate` function and the rotation direction was indicated by the sign of the count. However, given that there are two functions, the consensus of the committee was that reversing the direction of rotation with a negative value was confusing and unnecessary. The `stdc_rotate(unsigned-integer-type value, signed-integer-type count)` approach that chooses left / right based on the value also imposes the worst code generation properties of all the options. When using entirely runtime values, unless you deliberately have a variable-rotate/shift instruction, you are required to emit a branch to handle the two cases.

A more significant problem is the maximum number of bits that may be rotated. The types that can currently be rotated by the type generic functions are:

— a standard unsigned integer type, excluding bool;

— an extended unsigned integer type;

— or, a bit-precise unsigned integer type whose width matches any standard or extended integer type, excluding bool.

When rotating a value by more than its width, the rotation effectively wraps around. The bits shifted out of one end are re-introduced at the other end.

A rotation by, for example, 32 bits on a 32-bit integer is the same as no rotation at all.

The number of bits to rotate is taken modulo the bit width. This ensures that the rotation amount is always within the valid range for the specific bit width.

For example, if you have an 8-bit value and rotate it by 9 bits, it's the same as rotating it by 1 bit (9 mod 8 = 1).

Because the rotate functions are modulo the bit width, the count argument only needs to represent the width of the value. Similar to all the non-generic rotate functions, an `unsigned int` should be more than adequate in all cases. Consequently, there is no need to have a ***generic_count_type*** at all, when an `unsigned int` will do.

The ***generic_count_type*** parameter introduces a new undefined behavior when a negative count is passed. Paragraph 4 of subclause “7.18.17 Rotate Left” and “7.18.18 Rotate Right” both state:

> The ***generic_count_type*** `count` argument shall be a non-negative value of signed or unsigned integer type, or char.

Introducing a new undefined behavior is counterproductive while we are going through an effort to reduce the number of undefined behaviors in the standard, and completely unnecessary in this case.

As with most C functions, passing signed integers with negative values frequently indicates an error and can be detected using the `-Wsign-conversion` or similar flag. If the signed integer value is constrained to be within a valid range for the function, it can be cast to the unsigned integer type. It is well-defined behavior to convert a negative integer value to an unsigned integer type, but this usage is suspicious and should be corrected if it is a defect and documented if it is not. Passing a floating point value or a wider type is similarly suspect.

2 Proposal

This paper proposes replacing the ***generic_count_type*** parameter in the `stdc_rotate_left` and `stdc_rotate_right` functions with a fixed, unsigned integer parameter.

The proposed text uses a fixed, unsigned integer parameter of type `unsigned int`. Other proposals for using a fixed, unsigned integer parameter include:

- Use `unsigned long` (see message 30954)
- Use `uint_least32_t` (see message 30979)
- Use `size_t` (see message 31001)
- Use `count_t` (see message 31270)
- Use `width_t` (see message 31295)

Any of these solutions could be easily implemented as homework.

3 Proposed Text

Text in green is added to the C2Y working draft n3467. ~~Text in red~~ that has been struck through is removed from the C2Y working draft n3467.

Modify subclause "7.18.17 Rotate Left", paragraph 1:

```
#include <stdbit.h>
unsigned char stdc_rotate_left_uc(unsigned char value, unsigned int count);
unsigned short stdc_rotate_left_us(unsigned short value, unsigned int count);
unsigned int stdc_rotate_left_ui(unsigned int value, unsigned int count);
unsigned long stdc_rotate_left_ul(unsigned long value, unsigned int count);
unsigned long long stdc_rotate_left_ull(unsigned long long value,
  unsigned int count);
generic_value_type stdc_rotate_left(
  generic_value_type value, ~~generic_count_type~~ unsigned int count);
```

Modify subclause "7.18.17 Rotate Left", paragraph 4:

The type-generic function (marked by its generic_value_type argument) returns the above described result for a given input value so long as the generic_value_type is:
— a standard unsigned integer type, excluding bool;
— an extended unsigned integer type;

— or, a bit-precise unsigned integer type whose width matches any standard or extended integer type, excluding bool.

~~The generic_count_type count argument shall be a non-negative value of signed or unsigned integer type, or char.~~

Modify subclause "7.18.18 Rotate Right", paragraph 1:

```
#include <stdbit.h>
unsigned char stdc_rotate_right_uc(unsigned char value, unsigned int count);
unsigned short stdc_rotate_right_us(unsigned short value, unsigned int count);
unsigned int stdc_rotate_right_ui(unsigned int value, unsigned int count);
unsigned long stdc_rotate_right_ul(unsigned long value, unsigned int count);
unsigned long long stdc_rotate_right_ull(unsigned long long value,
  unsigned int count);
generic_value_type stdc_rotate_right(
  generic_value_type value, generic_count_type unsigned int count);
```

Modify subclause "7.18.18 Rotate Right", paragraph 4:

The type-generic function (marked by its generic_value_type argument) returns the above described result for a given input value so long as the generic_value_type is:
— a standard unsigned integer type, excluding bool;
— an extended unsigned integer type;
— or, a bit-precise unsigned integer type whose width matches any standard or extended integer type, excluding bool.

~~The generic_count_type count argument shall be a non-negative value of signed or unsigned integer type, or char.~~

4 Prior Art

These functions were added to C2Y.

5 Acknowledgements

We would like to recognize the following people for their help with this work: JeanHeyd Meneide, Aaron Ballman, Joseph Myers, Nevin Liber, Alejandro Colomar, Jakub Łukasiewicz, Jens Gustedt, Christopher Bazley, and Martin Uecker.