



Linux Ubuntu 26.04 Manual Pages on command 'static_assert.3'

\$ man static_assert.3

static_assert(3) Library Functions Manual static_assert(3)

NAME

static_assert, _Static_assert - fail compilation if assertion is false

LIBRARY

Standard C library (libc)

SYNOPSIS

```
#include <assert.h>

void static_assert(bool constant-expression, const char *msg);

/* Since C23: */

void static_assert(bool constant-expression);
```

DESCRIPTION

This macro is similar to `assert(3)`, but it works at compile time, generating a compilation error (with an optional message) when the input is false (i.e., compares equal to zero).

If the input is nonzero, no code is emitted.

`msg` must be a string literal. Since C23, this argument is optional.

There's a keyword, `_Static_assert()`, that behaves identically, and can be used without including `<assert.h>`.

RETURN VALUE

No value is returned.

VERSIONS

In C11, the second argument (`msg`) was mandatory; since C23, it can be omitted.

STANDARDS

C11 and later.

EXAMPLES

`static_assert()` can't be used in some places, like for example at global scope. For that, a macro `must_be()` can be written in terms of `static_assert()`. The following program uses the macro to get the size of an array safely.

```
#include <assert.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

/*
 * This macro behaves like static_assert(), failing to
 * compile if its argument is not true. However, it always
 * returns 0, which allows using it everywhere an expression
 * can be used.
 */

#define must_be(e) \
( \
    0 * (int) sizeof( \
        struct { \
            static_assert(e); \
            int ISO_C_forbids_a_struct_with_no_members; \
        } \
    ) \
)

#define is_same_type(a, b) \
    __builtin_types_compatible_p(typeof(a), typeof(b))

#define is_array(arr)    (!is_same_type((arr), &*(arr)))

#define must_be_array(arr) must_be(is_array(arr))

#define sizeof_array(arr) (sizeof(arr) + must_be_array(arr))

#define NITEMS(arr)      (sizeof((arr)) / sizeof((arr)[0]) \
                          + must_be_array(arr))

int    foo[10];

int8_t bar[sizeof_array(foo)];
```

```
int
main(void)
{
    for (size_t i = 0; i < NITEMS(foo); i++) {
        foo[i] = i;
    }
    memcpy(bar, foo, sizeof_array(bar));
    for (size_t i = 0; i < NITEMS(bar); i++) {
        printf("%d,", bar[i]);
    }
    exit(EXIT_SUCCESS);
}
```

SEE ALSO

[assert\(3\)](#)

[Linux man-pages 6.17](#)

2025-09-20

[static_assert\(3\)](#)