NAME

assert, **static_assert** - expression verification macro

SYNOPSIS

#include <assert.h>

assert(*expression*);

static_assert(*expression*);

static_assert(*expression*, *message*);

DESCRIPTION

The **assert**() macro tests the given *expression* and if it is false, the calling process is terminated. A diagnostic message is written to stderr and the function abort(3) is called, effectively terminating the program.

If *expression* is true, the **assert**() macro does nothing.

The **assert**() macro may be removed at compile time by defining NDEBUG as a macro (e.g., by using the cc(1) option **-D**NDEBUG). Unlike most other include files, <*assert.h*> may be included multiple times. Each time whether or not NDEBUG is defined determines the behavior of assert from that point forward until the end of the unit or another include of <*assert.h*>.

The **assert**() macro should only be used for ensuring the developer's expectations hold true. It is not appropriate for regular run-time error detection.

The **static_assert**() macro expands to **_Static_assert**(), and, contrarily to **assert**(), makes assertions at compile-time. Once the constraint is violated, the compiler produces a diagnostic message including the string literal message, if provided. The initial form of the **_Static_assert**() containing a string literal message was introduced in C11 standard, and the other form with no string literal is to be implemented by C2x and some compilers may lack its adoption at present.

EXAMPLES

The assertion:

```
assert(1 == 0);
```

generates a diagnostic message similar to the following:

```
Assertion failed: (1 == 0), function main, file main.c, line 100.
```

The following assert tries to assert there was no partial read:

```
assert(read(fd, buf, nbytes) == nbytes);
```

However, there are two problems. First, it checks for normal conditions, rather than conditions that indicate a bug. Second, the code will disappear if NDEBUG is defined, changing the semantics of the program.

The following asserts that the size of the S structure is 16. Otherwise, it produces a diagnostic message which points at the constraint and includes the provided string literal:

```
static_assert(sizeof(struct S) == 16, "size mismatch");
```

If none is provided, it only points at the constraint.

SEE ALSO

abort2(2), abort(3)

STANDARDS

The **assert**() macro conforms to ISO/IEC 9899:1999 ("ISO C99").

The **static_assert**() macro conforms to ISO/IEC 9899:2011 ("ISO C11").

HISTORY

An **assert** macro appeared in Version 7 AT&T UNIX.