assert

Defined in header `<cassert>`

```
#ifdef NDEBUG
#  define assert(condition) ((void)0)
#else
#  define assert(condition) /*implementation defined*/
#endif
```

The definition of the macro assert depends on another macro, `NDEBUG`, which is not defined by the standard library.

If `NDEBUG` is defined as a macro name at the point in the source code where `<cassert>` or `<assert.h>` is included, then assert does nothing.

If `NDEBUG` is not defined, then assert checks if its argument (which must have scalar type) compares equal to zero. If it does, assert outputs implementation-specific diagnostic information on the standard error output and calls std::abort. The diagnostic information is required to include the text of expression, as well as the values of the predefined variable `__func__` and (since C++11) the predefined macros `__FILE__` and `__LINE__`.

The expression `assert(E)` is guaranteed to be a constant subexpression, if either

- `NDEBUG` is defined at the point where assert is last defined or redefined (i.e., where the header `<cassert>` or `<assert.h>` was last included); or
- `E`, contextually converted to `bool`, is a constant subexpression that evaluates to `true`.

(since C++17)

Parameters

condition - expression of scalar type

Return value

(none)

Notes

Because assert is a function-like macro, commas anywhere in *condition* that are not protected by parentheses are interpreted as macro argument separators. Such commas are often found in template argument lists and list-initialization:

```
assert(std::is_same_v<int, int>); // error: assert does not take two arguments
assert((std::is_same_v<int, int>)); // OK: one argument
static_assert(std::is_same_v<int, int>); // OK: not a macro
std::complex<double> c;
assert(c == std::complex<double>{0, 0}); // error
assert((c == std::complex<double>{0, 0})); // OK
```

There is no standardized interface to add an additional message to assert errors. A portable way to include one is to use a comma operator provided it has not been overloaded:

```
assert(("There are five lights", 2 + 2 == 5));
```

The implementation of assert in Microsoft CRT (https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/assert-macro-assert-wassert?view=msvc-160) does not conform to C++11 and later revisions, because its underlying function (_wassert) takes neither `__func__` nor an equivalent replacement.

Example

Run this code

```
#include <iostream>
// uncomment to disable assert()
// #define NDEBUG
#include <cassert>
```

```
// Use (void) to silence unused warnings.
#define assertm(exp, msg) assert(((void)msg, exp))

int main()
{
    assert(2+2==4);
    std::cout << "Execution continues past the first assert\n";
    assertm(2+2==5, "There are five lights");
    std::cout << "Execution continues past the second assert\n";
    assert((2*2==4) && "Yet another way to add assert message");
}
```

Possible output:

```
Execution continues past the first assert
test: test.cc:10: int main(): Assertion `((void)"There are five lights", 2+2==5)' failed.
Aborted
```

See also

static_assert declaration(C++11)	performs compile-time assertion checking
abort	causes abnormal program termination (without cleaning up) (function)
C documentation for **assert**	

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/error/assert&oldid=138211"