

N

name space A collection of identifiers in which no two identifiers may have the same name. It is possible to use the same identifier for different purposes in the same scope (even though this might not be considered good style) provided each declaration of that identifier is in a different name space. Standard C defines the following name spaces for identifiers:

1. Formal parameter names in prototypes
2. Label names within a function
3. Structure, union, and enumeration tags
4. Each member set within a structure or union
5. All others—variables, functions, typedef names, and enumeration constants

Macro names are not included here because after preprocessing they no longer exist and, therefore, are not seen by the compiler. However, from a programmer's viewpoint, macro names should be thought of as sharing the same name space as all identifiers.

Each use of the identifier `x` in the following example is in a different name space. As such, the code fragment is valid.

```
void v(int x);
struct x {int x} s1;
struct y {int x} s2;

void f()
{
    int x;

    goto x;

x:    x = s1.x;
      x += s2.x;
}
```

namespace A C++ keyword that is not part of Standard C. If you think you might wish to move C code to a C++ environment in the future, you should refrain from using `namespace` as an identifier in new C code you write.

NAN^{C99} A macro, defined in `math.h`, that is defined iff the implementation supports quiet NaNs for the `float` type. It expands to a constant expression of type `float` whose value represents a quiet NaN.

NaN An encoding for a floating-point value signifying Not-a-Number.

`nan[f|l]`^{C99} A function that returns a quiet NaN, if they are supported, otherwise it returns zero.

```
#include <math.h>
double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

In the case of `nan`, the argument `tagp` is interpreted as follows:

```
nan("n-char-sequence") ≡ strtod("NaN(n-char-sequence)",
    (char**) NULL),
nan("") ≡ strtod("NaN()", (char**) NULL),
nan(any-other-string) ≡ strtod("NaN", (char**) NULL).
```

Calls to `nanf` and `nanl` are equivalent to the corresponding calls to `strtof` and `strtold`.

NaN, quiet A kind of NaN that propagates through almost every arithmetic operation without raising a floating-point exception.

NaN, signaling A kind of NaN that generally raises a floating-point exception when used as an arithmetic operand.

narrow type *See* type, narrow.

NCEG The Numerical C Extensions Group. Convened early in 1989 by Rex Jaeschke, NCEG's mission was to define extensions to C89 to help numerical programmers. The main areas investigated were aliasing, variable dimensioned arrays, array syntax, vector support, complex arithmetic, IEEE-754 issues, and exception handling. Eventually, NCEG became committee X3J11.1, which, ultimately, was absorbed into X3J11 and what is now J11. The resulting Technical Report (TR) made up the bulk of J11's contribution to C99.

NCITS/J11 The formal name for the U.S. standards committee J11.

NCITS/J16 The formal name for the U.S. standards committee J16.

`n_cs_precedes`^{C89} An `lconv` structure member that is a nonnegative number set to 1 or 0 if the `currency_symbol` respectively precedes or succeeds the value for a negative formatted monetary quantity. A value of `CHAR_MAX` indicates that the value is not available in the current locale. In the "C" locale this member must have the value `CHAR_MAX`.

`nearbyint[f|l]`^{C99} A function that rounds its argument to an integer value in floating-point format, using the current rounding direction and without raising the "inexact" floating-point exception.

`nearbyint[f|l]`

```
#include <math.h>
double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

nextafter[f|l]^{C99} A function that returns the next representable value after *x* in the direction of *y*, in the specified return type.

```
#include <math.h>
double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
```

See also **nexttoward**.

nexttoward[f|l]^{C99} A function that is equivalent to the corresponding version of the **nextafter** function except that **nexttoward**'s second parameter has type **long double** and that if *x* equals *y*, **nexttoward** returns *y* converted to the type of the function.

```
#include <math.h>
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

See also **nextafter**.

not^{C95} A macro, defined in `iso646.h`, that expands to the token `!`. It allows programmers using source character sets (such as ISO 646) that are missing certain characters necessary for writing C programs, to enter those characters using identifiers instead. Note that in C++, this name is a keyword.

not_eq^{C95} A macro, defined in `iso646.h`, that expands to the token `!=`. It allows programmers using source character sets (such as ISO 646) that are missing certain characters necessary for writing C programs, to enter those characters using identifiers instead. Note that in C++, this name is a keyword.

NDEBUG An option user-defined macro. If a definition for this macro is present at the point where `assert.h` is included, the `assert` macro takes on the definition `((void)0)`. If **NDEBUG** is not defined at the point of this header inclusion, the `assert` macro is defined to produce an assertion (which may result in abnormal program termination).

nearest integer functions The `math.h` functions `ceil`, `floor`, `lrint`, `lround`, `llround`, `nearbyint`, `rint`, and `round`, and their `float` and `long double` counterparts.

negative_sign^{C89} An `lconv` structure member that is a pointer to a string used to indicate a negative-valued formatted monetary quantity. If the string consists of "", the value is not available in the current locale or is of zero length. In the "C" locale this member must have the value "".

new A C++ keyword that is not part of Standard C. If you think you might wish to move C code to a C++ environment in the future, you should refrain from using `new` as an identifier in new C code you write.

new-line character One of the white space characters allowed in source text and as input to certain library functions. It is used to represent the logical end-of-line character, which is the effect generated when you type the ENTER or RETURN (or similar) key on a keyboard. On many systems, it is externally mapped into a carriage-return and line-feed pair, or just a line feed. However, internally, C always treats it as one character.

new-line escape sequence A new-line can be represented by the escape sequence `\n`.

noalias A type qualifier keyword invented by X3J11 and existed for a short while in a draft version of C89. The keyword was intended to provide the compiler with information about aliasing so that it could more aggressively optimize. While `noalias` is not part of Standard C, a subset of its properties was added in C99 via the keyword `restrict`.

nonlocal jumps header *See* `setjmp.h`.

not-equal-to operator *See* inequality operator.

n_sep_by_space^{C89} An `lconv` structure member that is a nonnegative number set to 1 or 0 if the `currency_symbol` respectively is or is not separated by a space from the value for a negative formatted monetary quantity. A value of `CHAR_MAX` indicates that the value is not available in the current locale. In the "C" locale this member must have the value `CHAR_MAX`.

n_sign_posn^{C89} An `lconv` structure member that is a nonnegative number set to a value indicating the positioning of the `negative_sign` for a negative formatted monetary quantity. The value is interpreted according to the following:

0 Parentheses surround the quantity and `currency_symbol`.

1 The sign string precedes the quantity and `currency_symbol`.

2 The sign string succeeds the quantity and `currency_symbol`.

3 The sign string immediately precedes the `currency_symbol`.

4 The sign string immediately succeeds the `currency_symbol`.

`CHAR_MAX` indicates that the value is not available in the current locale.

In the "C" locale this member must have the value `CHAR_MAX`.

NULL A macro that expands to an implementation-defined null pointer constant. It typically has a value of `0`, `0L`, or `((void *)0)` and is defined identically in each of the following headers: `locale.h`, `stddef.h`, `stdio.h`, `stdlib.h`, `string.h`, `time.h`, and `wchar.h`.

Standard C does not require the null pointer constant's value to have an internal representation of "all-bits-zero"—it can be any address guaranteed not to be used for an object or function.

`NULL` should never be used when a null character is meant because the definition of `NULL` is not always `0`.

null character The character with value `'\0'` used to terminate character strings. A string literal token (having the form `"ABCD"`) implicitly includes a trailing null character. Historically, many programmers used `NULL` in the context of a null character because they "knew" that `NULL` was defined as `0`. This has always been bad style and, with Standard C, actually can fail because `NULL` can be defined otherwise.

null pointer The result when a null pointer constant is converted to a pointer type. A comparison between the null pointer and a pointer to any object or function is guaranteed to be unequal.

null pointer constant *See* constant, null pointer.

null preprocessing directive # A preprocessing directive that has the form:

```
#
```

and has no effect. It is an historic artifact.

null statement A statement consisting solely of a semicolon.

null wide character^{C89} A wide character with code value zero. It can be written as `L'\0'`.

number classification macro^{C99} One of `FP_INFINITE`, `FP_NAN`, `FP_NORMAL`, `FP_SUBNORMAL`, and `FP_ZERO`, or other implementation-defined macros having names beginning with `FP_` and an uppercase letter, defined in `math.h`. These macros expand to integer constant expressions with distinct values.

Numerical C Extensions Group *See* NCEG.

numerical limits^{C89} *See* `float.h`; `limits.h`.

