

U

U suffix *See* constant, integer.

u suffix *See* constant, integer.

UCHAR_MAX^{C89} A macro, defined in `limits.h`, that designates the maximum value for an object of type `unsigned char`. It must be at least 255 (8 bits). This macro expands to an integer constant expression suitable for use with a `#if` directive. C99 requires `UCHAR_MAX` to be $2^{CHAR_BIT} - 1$.

UCN *See* universal character name.

uint_fast8_t^{C99} *See* `uint_fastN_t`.

uint_fast16_t^{C99} *See* `uint_fastN_t`.

uint_fast32_t^{C99} *See* `uint_fastN_t`.

uint_fast64_t^{C99} *See* `uint_fastN_t`.

UINT_FASTN_MAX^{C99} A macro, defined in `stdint.h`, that indicates the maximum value of the corresponding fastest minimum-width unsigned integer type, `uint_fastN_t`. It expands to an integer constant expression suitable for use with a `#if` directive.

uint_fastN_t^{C99} A type, defined in `stdint.h`, that is the fastest signed integer type with a width of at least N bits. For example, `uint_fast32_t` denotes a signed integer type with a width of at least 32 bits. The following types must be defined: `uint_fast8_t`, `uint_fast16_t`, `uint_fast32_t`, and `uint_fast64_t`. Other types of this form are optional. *See also* `int_fastN_t`; `UINT_FASTN_MAX`.

uint_least8_t^{C99} *See* `uint_leastN_t`.

uint_least16_t^{C99} *See* `uint_leastN_t`.

uint_least32_t^{C99} *See* `uint_leastN_t`.

uint_least64_t^{C99} *See* `uint_leastN_t`.

UINT_LEASTN_MAX^{C99} A macro, defined in `stdint.h`, that indicates the maximum value of the corresponding minimum-width unsigned integer type, `uint_leastN_t`. It expands to an integer constant expression suitable for use with a `#if` directive.

uint_leastN_t^{C99} A macro, defined in `stdint.h`, that expands to a signed integer type with a width of at least N bits, such that no signed integer type of lesser size has at least the specified width. For example, `uint_least32_t` denotes a signed integer type with a width of at

least 32 bits. The following types must be defined: `uint_least8_t`, `uint_least16_t`, `uint_least32_t`, and `uint_least64_t`. Other types of this form are optional.

See also `int_leastN_t`; `UINT_LEASTN_MAX`.

`UINT_MAX`^{C89} A macro, defined in `limits.h`, that designates the maximum value for an object of type `unsigned int`. It must be at least 65,535 (16 bits). This macro expands to an integer constant expression suitable for use with a `#if` directive.

`UINTMAX_C`^{C99} A function-like macro, defined in `stdint.h`, that has the form `UINTMAX_C(value)` and expands to an integer constant with the specified value and type `uintmax_t`. `value` is a decimal, octal, or hexadecimal constant whose value does not exceed the limits for the corresponding type..

`UINTMAX_MAX`^{C99} A macro, defined in `stdint.h`, that indicates the maximum value of `uintmax_t`, the corresponding greatest-width unsigned integer type. It expands to an integer constant expression suitable for use with a `#if` directive.

`uintmax_t`^{C99} A type, defined in `stdint.h`, that is an unsigned integer type capable of representing any value of any unsigned integer type, including extended types. Preprocessor arithmetic is done using this type. *See also* `intmax_t`; `UINTMAX_MAX`.

`UINTN_C`^{C99} A function-like macro, defined in `stdint.h`, that has the form `UINTN_C(value)` and expands to an unsigned integer constant with the specified value and type `uint_leastN_t`. `value` is a decimal, octal, or hexadecimal constant whose value does not exceed the limits for the corresponding type..

`UINTN_MAX`^{C99} A macro, defined in `stdint.h`, that indicates the maximum value of the corresponding exact-width unsigned integer type, `uintN_t`. It expands to an integer constant expression suitable for use with a `#if` directive.

`uintN_t`^{C99} A type, defined in `stdint.h`, that designates an unsigned integer type having width `N`. For example, `uint32_t` denotes an unsigned integer type with a width of exactly 32 bits.

Such types are optional, but if an implementation provides integer types with widths of 8, 16, 32, or 64 bits, it must define the corresponding typedef names. *See also* `UINTN_MAX`; `intN_t`.

`UINTPTR_MAX`^{C99} A macro, defined in `stdint.h`, that indicates the maximum value of `uintptr_t`, the corresponding pointer-holding unsigned integer type. It expands to an integer constant expression suitable for use with a `#if` directive.

uintptr_t^{C99} An optional type, defined in `stdint.h`, that designates an unsigned integer type such that any valid pointer to `void` can be converted to this type and back again, with the result comparing equal to the original pointer. *See also* `intptr_t`; `UINTPTR_MAX`.

ULLONG_MAX^{C99} A macro, defined in `limits.h`, that designates the maximum value for an object of type `unsigned long long int`. Its value must be at least 18,446,744,073,709,551,615 (64 bits). This macro expands to an integer constant expression suitable for use with a `#if` directive.

ULONG_MAX^{C89} A macro, defined in `limits.h`, that designates the maximum value for an object of type `unsigned long int`. It must be at least 4,294,967,295 (32 bits). This macro expands to an integer constant expression suitable for use with a `#if` directive.

unary arithmetic operators The unary operators `+`, `-`, `~`, and `!`.

unary minus operator A unary operator, `-`, that negates the value of its operand. The operand must have arithmetic type. Note there is no such thing as a negative constant in C. The expression `-32768` consists of two tokens—the unary minus and the constant `32768`. (Note that on a 16-bit, twos-complement machine, although `-32,768` is the smallest negative value one can store in an `int` object, the constant `-32768` actually has type `long int`. *See also* `constant`, `type of`.) The usual arithmetic conversions are performed on the operand, and the result has the promoted type. This operator associates right to left.

unary operator *See* `operator`, `unary`.

unary plus operator^{C89} A unary operator, `+`, that has no effect on the value of its operand. The operand must have arithmetic type. The usual arithmetic conversions are performed on the operand, and the result has the promoted type. This operator associates right to left.

#undef A preprocessor directive used to remove a macro definition. It is not an error to `#undef` a non-existent macro. It is used as follows:

```
#undef identifier
```

One reason to use this directive is to remove a macro to get at an underlying function version; for example,

```
#include <ctype.h>
#undef isalpha

int (*fp)(int) = isalpha;
```

#undef

Because library functions such as `isalpha` also can be implemented as macros, it is necessary to remove any such macro definition before taking the address of the underlying function. Unless this is done, a compilation error may result because if a macro version exists, it almost certainly won't be able to have its address taken.

It is not possible to `#undef` a predefined macro or the preprocessor operator `defined`.

undefined behavior Behavior for an erroneous or nonportable program construct or erroneous data for which the standard imposes no requirements. Examples include the passing of an argument of the incorrect type to a function and using the value of a function that does not return one.

underscore A character allowed in identifier names. Certain identifier names with leading underscores are reserved for use by implementers and future versions of Standard C. To avoid conflicts with private names invented by implementers in standard headers, never invent an identifier name that begins with an underscore. Some environments do not have the underscore character in their external character set. In such cases, it is typically mapped to some other character. The underscore is neither a punctuation nor an operator.

ungetc A function that pushes the character specified by `c` (after converting it to `unsigned char`) back into the input stream pointed to by `stream`.

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

Characters pushed back can be gotten (in the reverse order of which they were pushed back) with subsequent reads from that stream. If you call `rewind`, `fseek`, or `fsetpos`, any unread, pushed-back characters are discarded. Standard C guarantees only one character of pushback. If more characters are pushed back than can be handled, an error is returned.

If `ungetc` is successful, it returns `c`, and the end-of-file indicator for that stream is cleared; otherwise, it returns `EOF`. *See also* `ungetwc`.

ungetwc^{C95} A function that pushes the wide character specified by `c` (after converting it to `wint_t`) back into the input stream pointed to by `stream`.

```
#include <stdio.h>
#include <wchar.h>
wint_t ungetwc(wint_t c, FILE *stream);
```

Characters pushed back can be gotten (in the reverse order of which they were pushed back) with subsequent reads from that stream. If you call `rewind`, `fseek`, or `fsetpos`, any unread, pushed-back characters are discarded. Standard C guarantees only one character of pushback. If more characters are pushed back than can be handled, an error is returned.

If `ungetc` is successful, it returns `c`, and the end-of-file indicator for that stream is cleared; otherwise, it returns `WEOF`. *See also* `ungetc`.

Unicode A 16-bit character set developed by the Unicode Consortium. It was adopted as standard ISO/IEC 10646. New operating systems, new releases of some older systems, and some languages (most notably Java) are adopting Unicode as their basic character set. *See also* ISO/IEC 10646.UCS-2; `_STDC_ISO_10646_`; `wchar_t`; wide character.

union A union contains one or more members. Sufficient storage is allocated to a union so that, at any time, one occurrence of only one member can be stored. The size of a union is at least the size of its largest member. Although a union is syntactically very similar to a structure, it is subtly different. A union is defined using the keyword `union`. A union is neither a scalar nor an aggregate; it is in a category by itself.

If a union contains two or more structures that share a common initial sequence, and if an object of that type currently contains one of these structures, the common initial parts of those structures overlap directly, so they can be accessed via any of the structures.

union The keyword used to define a union type and to declare identifiers of that type; for example,

```
union tag {
    int i;
    double d;
    char *pc;
} u1;

union tag u2, *pu;
```

It is the programmer's responsibility to keep track of which member a union was last stored through. If a union is stored through one member and its value is retrieved through a member of different type, the result is undefined.

union initialization The use of an initializer in a union declaration. Prior to C89, unions could not have initializers. Standard C now permits this and uses an initializer format like that for structures. That is, the

initializer is a brace-delimited list where the list contains one expression unless the member of the element being initialized is an aggregate. The initializer is interpreted according to the first member of the union, which means that for the purpose of initialization only, it is significant which member is defined first. For example,

```
union {
    int i;
    double d;
} u = {1.2};
```

results in `u.i` being initialized with the integer value 1. If, however, the two members' order was swapped, `u.d` would be initialized with the floating-point value 1.2.

With the addition of designated initializers in C99, a union can be initialized using any of its members; for example,

```
union {
    // order of members i and j is irrelevant
} u = {.d = 1.2};
```

causes the union to be initialized through the `double` member `d`.

union tag *See* tag.

universal character name (UCN) A sequence of characters that can be used in identifiers, character constants, and string literals to represent characters not found in the basic character set. There are two forms: `\unnnn` and `\Unnnnnnnnn`, where *n* is a hexadecimal digit, allowing for the use of both 16- and 32-bit character sets. The coding scheme used is ISO 10646; therefore, for example, `\u03A0` and `\u03C0` represent the Greek letters Π and π , respectively, while `\u2020` and `\u2021` represent \dagger and \ddagger , respectively.

It is system and locale dependent whether an implementation will be able to convert any given UCN to a character in the execution character set.

Universal Time Coordinated The international standard name for what was previously known as Greenwich Mean Time (GMT). It is often abbreviated as UTC.

unordered A situation in which two floating-point numbers are not equal and neither number is greater than the other. A NaN is unordered when compared to any numeric value or to another NaN. *See* `isunordered`.

unqualified type^{C89} *See* type, unqualified.

unsigned A keyword used as an unsigned integer data type prefix. It may be applied to `char`, `short int`, `int`, `long int`, and `long long int`. It allows unsigned arithmetic to be performed. When used on its own, it implies `unsigned int`. *See also* integer type.

unsigned int suffix *See* constant, integer.

unsigned integer types The types `unsigned char`, `unsigned short`, `unsigned int`, `unsigned long`, and `unsigned long long`. *See also* bit-field, plain int; `char`, plain.

unsigned long A permitted abbreviation for `unsigned long int`.

unsigned long int A standard integer type. Standard C requires it to be at least 32 bits. *See also* integer type.

unsigned long long^{C99} A permitted abbreviation for `unsigned long long int`.

unsigned long long int^{C99} A standard integer type. Standard C requires it to be at least 62 bits. Note that an unsuffixed decimal constant cannot have this type. *See also* integer type.

unsigned preserving rule The rule by which expressions having type `unsigned int` bit-field, `unsigned char`, or `unsigned short`, are converted to `unsigned int` when these “narrow” types are widened. This rule was widely followed prior to the advent of Standard C, which now requires following the value preserving rule instead. *See also* conversion, integer type; value-preserving rule.

unsigned type conversion *See* conversion, integer type.

unspecified behavior Behavior for a correct program construct and correct data for which the standard imposes no requirements. For example, the order of evaluation of expressions (except those involving `&&`, `||`, `?:` and the comma operator) is unspecified.

unspecified value A valid value of the appropriate type where the C standard imposes no requirements on which value is used.

UP rule *See* unsigned preserving rule.

USHRT_MAX^{C89} A macro, defined in `limits.h`, that designates the maximum value for an object of type `unsigned short int`. It must be at least 65,535. This macro expands to an integer constant expression suitable for use with a `#if` directive.

using 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 `using` as an identifier in new C code you write.

usual arithmetic conversion *See* conversion, usual arithmetic.

UTC *See* Universal Time Coordinated.

