

A

abort A function that causes the program to terminate abnormally unless the signal `SIGABRT` is being caught and the signal handler does not return. It does not return to its caller.

```
#include <stdlib.h>
void abort(void);
```

It is implementation defined as to whether or not output streams are flushed, open streams are closed, or temporary files are removed. The exit code of the program is some implementation-defined value that represents failure. A program abort also can be generated by a call to `raise` using the argument `SIGABRT`. *See also* `assert`; `exit`.

abs A function that computes the absolute value of its argument `j`.

```
#include <stdlib.h>
int abs(int j);
```

The behavior is undefined if the result cannot be represented (such as `abs(x)` on a two's-complement 16-bit machine, where `x` has the value `-32,768`).

absolute-value functions A family of functions, each member of which computes the absolute value of its argument. The floating-point members (`fabs`, `fabsf`, and `fabsl`) are declared in `math.h`, the integer members (`abs`, `labs`, and `llabs`) are declared in `stdlib.h`, and the complex members (`cabs`, `cabsf`, and `cabsl`) are declared in `complex.h`.

abstract machine A theoretical system on which a language standard is modeled. In the abstract machine, optimization (and hence, the semantics of `volatile`) is irrelevant. However, this is not so in the real world, and things such as sequence points are needed so optimizers know how much flexibility they have.

access To read or modify the value of an object at run time.

acos[f|l] A function that computes the arc cosine of its argument `x` and returns a value in the range $[0, \pi]$ radians.

```
#include <math.h>
double acos(double x);
float acosf(float x);
long double acosl(long double x);
```

If the argument is not in the range $[-1, +1]$, a domain error occurs.

The `float` and `long double` versions were an invention of C89, where they were optional; however, in C99, they are required.

`acosh[f|l]`^{C99} A function that computes the (nonnegative) arc hyperbolic cosine of its argument `x` and returns a value in the range $[0, +\infty]$.

```
#include <math.h>
double acosh(double x);
float acoshf(float x);
long double acoshl(long double x);
```

If the argument is less than 1, a domain error occurs.

active position The position on a display device at which the next character output by `fputc` will appear. In most Western cultures, screen output is produced from left to right and top to bottom. However, in other environments this is not the case. The direction used for such writing is locale specific.

addition assignment operator A binary operator, `+=`, that permits addition and assignment to be combined such that `exp1 += exp2` is equivalent to `exp1 = exp1 + exp2` except that in the former, `exp1` is only evaluated once. The order of evaluation of the operands is unspecified. Both operands must either have arithmetic type, or the right may have integer type if the left is a pointer to an object. The left operand must be a modifiable lvalue. The type of the result is the type of `exp1`. This operator associates right to left. *See also* assignment operator, compound.

addition operator A binary operator, `+`, that causes the values of its operands to be added together. The order of evaluation of the two operands is unspecified. Both operands must either have arithmetic type, or one may have integer type if the other is a pointer to an object. The usual arithmetic conversions are performed on the operands. This operator associates left to right.

address The memory location at which an object or function resides. Most machines are either byte- or word-addressable. That is, the level of granularity for unique addresses is a byte or word, respectively. An object or function can (and often does) occupy more than one memory location, in which case, the address of that object or function is the address of its beginning. In C, an expression designating an address is also called a pointer expression or, more simply, a pointer. *See also* address-of operator.

address-of operator A unary operator, `&`, that causes the address of its operand to be taken. The operand must be either a function designator

or an lvalue that designates an object (other than a bit-field or object declared with storage class `register`). This operator associates right to left. The result has type “pointer to the type of the operand.” Note that if `ary` designates an array, `&ary` is a pointer to the whole array, which is quite different from `&ary[0]`, a pointer to the first element in the array.

aggregate An array or structure object. Typically, an aggregate contains two or more elements. However, C does permit an array of one element and a structure with only one member. Nonaggregate object types are scalars and unions.

alert escape sequence^{C89} An escape sequence, `\a`, which indicates an alert (such as a terminal bell).

aliasing A method of giving something an alternate name. C permits an infinite number of aliases for an object or function by using pointers. You can access an object or function either by its name or via any one of the pointers that points directly or indirectly to it. Historically, there has been no way to promise a function that has arrays passed to it as arguments, that those arguments don’t overlap, or that in general two pointers point to distinct objects. As such, it was difficult to optimize certain code and impossible to vectorize or parallelize loops if overlap potentially existed. *See* type, effective.

During the deliberations for C89, the standards committee invented the `noalias` type qualifier to help resolve this long-standing and difficult issue. However, after considerable and heated debate, that keyword was removed from the working draft. After further research by the NCEG, C99 added the qualifier `restrict` to help address this issue.

alignment The process of placing objects of certain types in memory such that their storage begins on specific address boundaries. Some machines require objects to be aligned and, while other machines can manipulate objects on any boundary, it is more efficient if words are aligned on word boundaries, longwords on longword boundaries, and so on. Structures may contain holes between members (or after the last member) so padding can be provided by the compiler to honor alignment requirements. Alignment might either be forced on you by your compiler or it might be user selectable via a compiler option. Some compilers allow alignment to be specified using a pragma (called `pack`, for example).

Amendment 1^{C95} *See* C95.

and^{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.

AND assignment operator, bitwise A binary operator, `&=`, that permits bitwise AND and assignment to be combined such that `exp1 &= exp2` is equivalent to `exp1 = exp1 & exp2` except that in the former, `exp1` is evaluated only once. Both operands must have integer type. The order of evaluation of the operands is unspecified. The left operand must be a modifiable lvalue. The type of the result is the type of `exp1`. This operator associates right to left. *See also* assignment operator, compound.

and_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.

AND operator, bitwise A binary operator, `&`, that performs a bitwise AND of its operands. Both operands must have integer type. The order of evaluation of the operands is unspecified. The usual arithmetic conversions are performed on the operands. This operator associates left to right.

AND operator, logical A binary operator, `&&`, that performs a logical AND of its operands. Both operands must have scalar type. The result has type `int` and value 0 (if false) or 1 (if true). There is a sequence point after the evaluation of the left operand. If the left operand tests false, the right operand is not evaluated. This operator associates left to right.

ANSI The acronym for the American National Standards Institute, the U.S. national standards body responsible for most programming language standards.

ANSI C The formal definition of the C language, preprocessor, and runtime library as accepted by the American National Standards Institute (ANSI). The first version is known as C89, was produced by committee X3J11, and was officially called ANSI X3.159-1989. The second version (which is technically identical to the first) is known as C90 and was produced by the ISO C committee SC 22/WG14 and then adopted by ANSI as ANSI/ISO 9899-1990. Later, an amendment, known as C95, was published. The third version, C99, was produced by committees SC 22/WG14 and NCITS/J11; its formal designation is ANSI/ISO 9899-1999. Synonymous with Standard C.

ANSI/IEEE 754 *See* IEEE Floating-Point Arithmetic Standards.

ANSI/IEEE 854 *See* IEEE Floating-Point Arithmetic Standards.

argc The first argument passed to `main`. This `int` argument contains the number of arguments found on the command line when the program was

invoked. Typically its value is at least one, although the actual name used on the command line to invoke the program itself might not be preserved by the task loader. Note that while the name `argc` is widely used for this argument, the name `argc` is local to that function and, in fact, can be replaced by any unreserved identifier the programmer chooses. *See also* `argv`; `envp`.

argument *See* argument, actual; argument, formal.

argument, actual The expression actually passed in a call to a function, or the token sequence actually passed in a call to a function-like macro. For example, in `f(a, b + c)`, `a` and `b + c` are the two actual arguments passed to `f`. An actual argument may contain commas, provided they are enclosed in parentheses. *See also* argument, formal.

argument, command-line One of a (possible) set of tokens specified on the command line when a program is invoked. By definition, each command-line argument is separated by horizontal white space. The number and value of command-line arguments are accessed via `argc` and `argv`, respectively.

argument, formal The object declared in a function definition that takes on a value when that function is called, or an identifier in the comma-separated list in a function-like macro definition. For example, in

```
void f(int a1, long a2) { /* ... */ }

#define Add(a, b) (a + b)
```

`a1` and `a2` are the formal arguments expected by the function `f`, and `a` and `b` are the formal arguments expected by the macro `Add`. Synonymous with parameter. *See also* argument, actual.

argument list A comma-separated list of actual or formal arguments.

argument list, variable An argument list containing at least one argument and having a trailing ellipsis. The only Standard C functions having variable argument lists are the `printf/wprintf` and `scanf/wscanf` families. *See also* `stdarg.h` for details of defining a function that expects a variable argument list. Prior to C99, macros could not be defined or called with variable argument lists; now they can by using `_VA_ARGS_`.

argument promotion, default *See* conversion, function arguments.

argv The second argument passed to `main`. `argv` is an array of pointers to `char`, each of which points to a string specified as a command-line argument. The number of elements in the array is `argc+1` where `argv[argc]` is `NULL`. If the program's name is not available, `argv[0]` points to a null

character. Note that different loaders/compiler may treat quoted arguments and arguments with embedded white space differently. Also, while the name `argv` is widely used for the second argument to `main`, the name `argv` is local to that function and, in fact, can be replaced by any unreserved identifier the programmer chooses. *See also* `argc`; `envp`.

arithmetic conversions, usual *See* conversion, usual arithmetic.

arithmetic type Any one of the integer or floating types.

array An aggregate type consisting of one or more elements, each of which has exactly the same attributes and occupies consecutively higher memory locations. The number of dimensions in an array is not restricted by C except that an implementation is required to support at least 12. The theoretical maximum value for the size of any dimension of an array is the maximum value of the type `size_t`. Elements in each dimension begin at subscript zero and each dimension is specified separately in both the array declaration and in subscript expressions. *See* array, variable-length.

array, storage order Row-major order. That is, arrays are stored such that the right-most subscript varies fastest.

array, variable-length^{C99} An array in which the size of any dimension can be a nonconstant integer expression, provided that the array is declared with block scope and automatic storage duration, or prototype scope. The following example includes some variable-length arrays (VLAs):

```
void f(int m, int C[m][m])
{
    int v1[m];
    double v2[6][m];
    long int v3[m][4][n];
    typedef int VLA[m][m];
    /* ... */
}
```

In a function declarator that is not also the definition of that function, parameters may use the notation `[*]` to indicate variable-length array types; for example,

```
double maximum(int n, int m, double a[*][*]);
```

array, zero-sized An array containing no elements. Prior to C89, quite a few implementations allowed arrays with zero elements. Such an array was typically used as the last member of a structure. Support for this was added in C99; it is known as a flexible array member.

Standard C allows an implementation to accept an allocation request for zero bytes via `calloc` and `malloc`, thereby allowing the creation of an array with zero elements.

arrow operator A binary operator, `->`, that is used to select the right operand from the structure or union pointed to by the left operand. The order of evaluation of the operands is unspecified. The type of the left operand must be pointer to structure or union and the right operand must be the name of a member in that structure or union. This operator always produces an lvalue. A `->` expression can always be rewritten using the dot operator. For example, `p->m` is equivalent to `(*p).m`. The type of the result is the type of the named member. This operator associates left to right.

“as if” rule A characteristic of C set up in the C standard that permits implementations to take full advantage of their environment without being unduly forced to perform inefficient operations. For example, the traditional requirement that all `char` arithmetic be done in `int` precision, is no longer true from a practical viewpoint. Now, an implementation is permitted to perform optimizations and operations in any way it sees fit, provided it arrives at the same result *as if* it had followed the strict rules of the abstract machine for which the standard was defined. That is, if there is no discernable difference between doing an optimization and not doing it, the optimization is permitted.

ASCII An acronym for the American Standard Code for Information Interchange. This 7-bit code can represent 128 different characters and is in common use. Standard C is not character-set specific. *See also* EBCDIC; `isctrl`; Unicode.

asctime A function that converts the broken-down time stored in the structure pointed to by `timeptr` into a string.

```
#include <time.h>
char *asctime(const struct tm *timeptr);
```

The pointer returned points to a string containing the time in the form `Day Mon dd hh:mm:ss yyyy\n\0`. This function is not locale-specific. For locale-specific times use `strftime`. (The `asc` prefix implies ASCII and is purely historic—Standard C is not character-set specific.)

asin[f|l] A function that computes the arc sine of its argument `x` and returns a value in the range $[-\pi/2, +\pi/2]$ radians.

```
#include <math.h>
double asin(double x);
float asinf(float x);
long double asinl(long double x);
```

If the argument is not in the range $[-1, +1]$, a domain error occurs.

The `float` and `long double` versions were an invention of C89, where they were optional; however, in C99, they are required.

asinh[f|l]^{C99} A function that computes the arc hyperbolic sine of its argument `x`.

```
#include <math.h>
double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

asm Numerous compilers permit assembler instructions to be inserted inline in C source. They typically do this using either a keyword such as `asm` or a preprocessor directive such as `#asm` possibly with a matching `#endasm`. This is not part of Standard C.

`asm` is also a C++ keyword. If you think you might wish to move C code to a C++ environment in the future, you should refrain from using `asm` as an identifier in new C code you write.

assert A macro, defined in `assert.h`, that causes a diagnostic message to be written to `stderr` and then `abort` to be called, provided the macro argument evaluates to false and the macro `NDEBUG` is not currently defined, for example,

```
#include <assert.h>
void assert(type exp);
```

Prior to C99, *type* was required to be `int`; however, C99 allows it to be any scalar type.

If the user has defined `NDEBUG` before `assert.h` is included, `assert` is defined as a `void` expression and all calls to `assert` become vacuous.

The format of the message output is implementation defined. However, it must contain the macro argument in its text form, with the source filename and line number of the invocation of the failing `assert`, and, in C99, the name of the enclosing function. *See also* `__FILE__`; `__func__`; `__LINE__`.

The behavior is undefined if you `#undef assert` to try to get a function version.

assert.h A header that provides a crude debugging traceback facility. It is the only standard header designed to be included multiple times such that it can behave differently each time it is included. *See also* **assert**. `assert.h` contains a definition for the following identifier:

assert Put diagnostic assertions in code

assignment The operation of assigning the value of an expression to the memory location designated by another expression (which is typically a variable name or an expression of the form `a[i]`, `*ptr`, `p->m`, or `s.m`). The expression designating the destination must be a modifiable lvalue. Because assignment is implemented in C using operators, assignment expressions have type and value and therefore can be embedded in larger expressions. *See also* assignment operator, compound; assignment operator, simple.

assignment operator, compound The general form of a binary compound assignment operator. `x op= y` is equivalent to `x = x op y` except that in the former, `x` is only evaluated once. The complete set of compound assignment operators is: `+=`, `-=`, `*=`, `/=`, `%=`, `>>=`, `<<=`, `&=`, `^=`, and `|=`. *See also* assignment operator, compound, archaic; assignment operator, simple.

assignment operator, compound, archaic An archaic form (`=op`) for a binary compound assignment operator was long ago (pre-C89) replaced by `op=` and is not part of Standard C.

assignment operator, simple A binary operator, `=`, that causes the value of the right operand to be stored in the location designated by the left operand. The order of evaluation of the two operands is unspecified. This operator associates right to left. The left operand must be a modifiable lvalue. *See also* assignment operator, compound. Note that this operator *must not* be confused with the equality operator `==`. The two are easily confused and can be interchanged syntactically; however, their semantics are quite different. For example, in the following:

```
if (a = b) /* ... */
```

the value of `b` is assigned to `a`, and the value of `a` is tested for truth. In the following:

```
if (a == b) /* ... */
```

the values of `a` and `b` are compared for equality.

associativity The property that determines the relative precedence of operators when the precedence table shows them as otherwise having the same precedence. Associativity is either left to right or right to left, as the expression is written. For example, $a/b/c$ is equivalent to $(a/b)/c$ because the associativity of the division operator is left to right. On the other hand, $!++i$ is equivalent to $(!(++i))$ because these unary operators associate right to left.

atan[f|l] A function that computes the arc tangent of its argument x .

```
#include <math.h>
double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

It returns a value in the range $[-\pi/2, +\pi/2]$ radians.

The `float` and `long double` versions were an invention of C89, where they were optional; however, in C99, they are required.

atan2[f|l] A function that computes the arc tangent of y/x and returns a value in the range $[-\pi, +\pi]$ radians.

```
#include <math.h>
double atan2(double y, double x);
float atan2f(float y, float x);
long double atan2l(long double y, long double x);
```

If both arguments are 0 a domain error may occur. The sign of both arguments is used to determine the quadrant of the returned value.

The `float` and `long double` versions were an invention of C89, where they were optional; however, in C99, they are required.

atanh[f|l]^{C99} A function that computes the arc hyperbolic tangent of its argument x .

```
#include <math.h>
double atanh(double x);
float atanhf(float x);
long double atanh1(long double x);
```

An argument outside the interval $[-1, +1]$ causes a domain error. An argument of ± 1 might cause a range error.

atexit^{C89} A function that permits a function to be registered such that it is called automatically by the implementation during normal program termination.

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

The function being registered must take no arguments and have no return value. Standard C requires that at least 32 functions can be registered. (However, to get around any limitations in this regard, you can always register just one function and have it call the others directly. This way, the other functions can also have argument lists and return values.) The same function can be registered more than once. If the registration fails, a nonzero value is returned; otherwise, zero is returned. *See also* `exit`; `_Exit`.

atof A function that converts the leading part of the string pointed to by `nptr`, to a double value.

```
#include <stdlib.h>
double atof(const char *nptr);
```

A call to `atof` is identical to

```
strtod(nptr, (char **)NULL)
```

except that `strtod` can handle errors. Note that the format of a valid floating-point value on input is locale specific. `strtod` is preferred over `atof` because the former provides more control over the conversion and error detection and handling.

atoi A function that converts the leading part of the string pointed to by `nptr` to an `int` value.

```
#include <stdlib.h>
int atoi(const char *nptr);
```

A call to `atoi` is identical to

```
(int) strtol(nptr, (char **)NULL, 10)
```

except that `strtol` can handle errors. `strtol` is preferred over `atoi` because the former provides more control over the conversion and error detection and handling.

atol A function that converts the leading part of the string pointed to by `nptr` to a long `int` value.

```
#include <stdlib.h>
long int atol(const char *nptr);
```

A call to `atol` is equivalent to

```
strtoul(nptr, (char **)NULL, 10)
```

except that `strtoul` can handle errors. `strtoul` is preferred over `atol` because the former provides more control over the conversion and error detection and handling.

`atoll`^{C99} A function that converts the leading part of the string pointed to by `nptr` to a long long int value.

```
#include <stdlib.h>
long long int atoll(const char *nptr);
```

A call to `atoll` is equivalent to

```
strtoll(nptr, (char **)NULL, 10)
```

except that `strtoll` can handle errors. `strtoll` is preferred over `atoll` because the former provides more control over the conversion and error detection and handling.

auto A storage class keyword used in the declaration of an object inside a function definition to designate automatic storage duration. If such a declaration contains no storage class keyword, `auto` is assumed. (As such, this keyword is rarely used and is *never* needed.) Formal parameters in a function definition behave as if they were declared as `auto` (unless they are explicitly declared with `register`). However, the keyword `auto` cannot actually be present in a formal parameter declaration.

automatic storage duration *See* storage duration, automatic.

