

R

radix point The point that separates the integer and fractional parts of a number. C always uses a period as the radix point character in floating point constants. In the standard "C" locale, a period is also used as the radix point in functions such as `printf`, `scanf`, and `strtod`. The standard routines using a radix point that is affected by the locale are: `atof`, `fprintf`, `fscanf`, `localeconv`, `printf`, `scanf`, `sprintf`, `sscanf`, `strtof`, `strtod`, `strtold`, `vfprintf`, `vprintf`, and `vsprintf`, and their wide-character counterparts.

raise A function that is used to send a signal `sig` to the program. It is useful for testing user-written signal handlers.

```
#include <signal.h>
int raise(int sig);
```

`raise` returns a value of zero if the raise was successful. Otherwise, it returns a nonzero value.

rand Repeated calls to this function generate a sequence of pseudo random integers in the range 0 to `RAND_MAX`. A single call generates a single number only.

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

See also `srand`.

RAND_MAX A macro, defined in `stdlib.h`, that expands to an integer constant expression representing the maximum value possible from `rand`. Standard C requires `RAND_MAX` to be at least 32,767. Its integer type is not specified, so use an explicit cast (or a prototype) when passing it as an argument.

range error An error that occurs if the result of the function cannot be represented in some specified type. If the result overflows, the function returns the value of `HUGE_VAL`, `HUGE_VALF`, or `HUGE_VALL`, as appropriate, with the same sign as the correct value would have. `errno` is set to the macro `ERANGE`. If the result underflows, the function returns 0 and `errno` may or may not be set to `ERANGE`, as the implementation defines. *See also* `math_errhandling`; `MATH_ERREXCEPT`; `MATH_ERRNO`.

realloc A function that changes the size of the dynamically allocated memory pointed to by `ptr` to size `size`. It returns the address of the (possibly) new location.

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

If `ptr` is `NULL`, `realloc` behaves like `malloc`. Otherwise, if `ptr` is not a value previously returned by `calloc`, `malloc`, or `realloc`, the behavior is undefined. The same is true if `ptr` points to space that has been freed. `size` is absolute, not relative. If `size` is larger than the size of the existing space, new uninitialized contiguous space is allocated at the end; the previous contents of the space are preserved. If `size` is smaller, the excess space is freed; however, the contents of the retained space are preserved. If `realloc` cannot allocate the requested space, `NULL` is returned and the contents of the space pointed to by `ptr` remain intact. If `ptr` is non-`NULL` and `size` is 0, `realloc` acts like `free`.

Whenever the size of space is changed by `realloc`, the new space may begin at an address different from that given it, even when `realloc` is truncating. Therefore, if you use `realloc` in this manner, you must be aware of pointers that point into this possibly moved space. For example, if you build a linked list there and use `realloc` to allocate more (or less) space for the chain, it is possible that the space will be moved, in which case, the pointers now point to where successive links used to be, not where they are now. You should always use `realloc` as follows:

```
p2 = realloc(p1, new_size);
if (p2 != NULL)
    p1 = p2;
```

This way, you never care whether the object has been relocated because you always update `p1` with each successful call to point to the (possibly new) location.

recursion A recursive function is one that invokes itself directly or indirectly. For each invocation, a new set of automatic objects (if any) is created. *See also* recursion.

redefinition of macro *See* benign redefinition; macro, redefinition of.

redirection characters UNIX and MS-DOS/Windows (and other operating systems) allow `stdin` and `stdout` to be redirected using the command-line symbols `<`, `>`, and `>>`. They also might provide a way to redirect `stderr`.

register A storage class keyword used in the declaration of an object inside a function definition to designate automatic storage duration. `register` is a hint to the compiler to place the object in some “fast” location of memory, such as a machine register. If the compiler cannot or chooses

not to do so, the keyword is treated as if it were `auto`. `register` also may be used in a prototype and in a function argument declaration.

An object with storage class `register` cannot have its address taken either explicitly (via the `&` operator) or implicitly (by passing an array to a function, for example). This helps an optimizer because it knows that object has no aliases.

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

rem^{C89} The name of one of the two members in the structure types `div_t`, `ldiv_t`, and `lldiv_t`. It is used to represent the remainder of an integer division and has type `int`, `long int`, and `long long int`, respectively, in these structures. *See also* `quot`.

remainder[f|l]^{C99} A function that returns remainder $x \text{ REM } y$, as required by IEC 60559.

```
#include <math.h>
double remainder(double x, double y);
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

According to C99, “When $y \neq 0$, the remainder $r = x \text{ REM } y$ is defined regardless of the rounding mode by the mathematical relation $r = x - ny$, where n is the integer nearest the exact value of x/y ; whenever $|n - x/y| = 1/2$, then n is even. Thus, the remainder is always exact. If $r = 0$, its sign shall be that of x .”

remainder assignment operator A binary operator, `%=`, that permits remainder 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. 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.

remainder functions `fmod`, `remquo`, and `remainder`, declared in `math.h`.

remainder operator A binary operator, `%`, that computes the remainder when its left operand is divided by its right. Both operands must have integer type. (To compute a floating-point remainder, use the `fmod`, `remquo`, or `remainder` library functions.) The usual arithmetic conversions are performed on the operands. The order of evaluation of the

operands is unspecified. This operator associates left to right. If either operand is negative, the sign of the remainder is implementation defined.

remove A function that causes the file whose name is pointed to by **filename** to be made inaccessible by that name.

```
#include <stdio.h>
int remove(const char *filename);
```

On many systems, the file is actually deleted. However, it may be that you are removing a synonym for a file's name, rather than deleting the file itself. In such cases, when the last synonym is being removed, the file is typically deleted. If **remove** succeeds, it returns a zero value; otherwise it returns a nonzero value.

remquo[f|l]^{C99} A function that computes the remainder and quotient of x/y .

```
#include <math.h>
double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y,
    int *quo);
```

rename A function that causes a file, currently known by the name pointed to by **old**, to be known by the name pointed to by **new**.

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

If a file called **new** already exists, the behavior is implementation defined. If **rename** succeeds, it returns a zero value; otherwise it returns a nonzero value. On failure, the file still has the name **old**.

replacement list, macro *See* macro.

reserved identifier *See* identifier, reserved.

restore calling environment function *See* longjmp.

restrict^{C99} A type qualifier that is intended to help generate optimal code by allowing the programmer to promise that all accesses to a given object are done through a particular pointer by making that pointer be **restrict**-qualified.

Consider the following declaration of **memcpy** in **string.h**:

```
void *memcpy(void * s1, const void * s2, size_t n);
```

By declaring both pointers to be `restrict`-qualified, we promise that the source and destination objects do not overlap, allowing a more efficient implementation.

Another good use of this keyword is in allocating memory; for example,

```
int * restrict p = malloc(100 * sizeof(int));
```

By declaring `p` to be `restrict`-qualified, the compiler can assume that no other pointers point to the same block of memory.

`restrict` may also be used inside any dimension of an array parameter (possibly along with `const` and/or `static`); for example,

```
void copy(int s[restrict], int d[restrict]);
```

This declaration specifies that the array arguments' do not overlap.

return A statement that causes control to be returned to the calling function. `return exp;` from `main` is equivalent to `exit(exp);`. A non-void function may return a value using the syntax `return exp;`. `return` is used as follows:

```
return [ expression ];
```

Falling through the end of the outermost closing brace in a function is an implied `return` without an expression. And if the function has a non-void return type, the value actually returned is undefined.

expression is a full expression.

rewind A function that sets the file's file position indicator to the start of the file.

```
#include <stdio.h>
void rewind(FILE *stream);
```

A call to `rewind` is identical to a call to `fseek` with `offset 0L` and `whence SEEK_SET`. However, `rewind` also clears the error indicator as well.

right-shift assignment operator A binary operator, `>>=`, that permits right shift 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. Both operands must have integer type, and the left operand must be a modifiable lvalue. The order of evaluation of

the operands is unspecified. The type of the result is the type of *exp1*. This operator associates right to left. If the value of the right operand is negative, or equal to or greater than the number of bits in the promoted left operand, the behavior is undefined. If the left operand has a signed type and negative value, the result is implementation defined. *See also* assignment operator, compound.

right-shift operator A binary operator, `>>`, that causes the value of its left operand to be shifted right by the number of bits specified by its right operand. Both operands must have integer type. The order of evaluation of the operands is unspecified. This operator associates left to right. The usual arithmetic conversions are performed on the operands. If the value of the right operand is negative, or equal to or greater than the number of bits in the promoted left operand, the behavior is undefined. If the left operand has a signed type and negative value, the result is implementation defined.

`rint[f|l]`^{C99} A function that returns the rounded integer value for its argument, using the current rounding direction.

```
#include <math.h>
double rint(double x);
float rintf(float x);
long double rintl(long double x);
```

This function differs from `nearbyint` in the raising of the “inexact” floating-point exception.

Ritchie, Dennis M. A Distinguished Member of the Bell Labs Computer Science Research Center. He is the principal designer of the C language and a major contributor to the UNIX operating system. He is the “R” in K&R.

`round[f|l]`^{C99} A function that returns the rounded integer value of its argument.

```
#include <math.h>
double round(double x);
float roundf(float x);
long double roundl(long double x);
```

The argument is rounded to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

rounding direction^{C99} An implementation may support one or more rounding directions by defining the macros `FE_DOWNWARD`, `FE_TONEAREST`,

FE_TOWARDZERO, and FE_UPWARD in fenv.h. They may also support other other rounding directions by providing macros whose names begin with FE_ and an uppercase letter. In any event, each defined rounding-direction macro must expand to an integer constant expression whose nonnegative value is distinct from all other such macros.

rounding mode *See* FLT_ROUNDS

rvalue The value of an expression. The name comes from the fact that an rvalue is often found on the right-hand side of assignments. *See also* lvalue.

