NAME

econvert, fconvert, gconvert, seconvert, sfconvert, sgconvert, qeconvert, qfconvert, qgconvert, ecvt, fcvt, gcvt – output conversion

SYNOPSIS

#include <floatingpoint.h>

char ∗**econvert(double** *value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign*, **char** ∗*buf***);**

char ∗**fconvert(double** *value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign*, **char** ∗*buf***);**

char ∗**gconvert(double** *value*, **int** *ndigit*, **int** *trailing*, **char** ∗*buf***);**

char ∗**seconvert(single** ∗*value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign*, **char** ∗*buf***);**

char ∗**sfconvert(single** ∗*value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign*, **char** ∗*buf***);**

char ∗**sgconvert(single** ∗*value*, **int** *ndigit*, **int** *trailing*, **char** ∗*buf);*

char ∗**qeconvert(quadruple** ∗*value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign*, **char** ∗*buf***);**

char ∗**qfconvert(quadruple** ∗*value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign*, **char** ∗*buf***);**

char ∗**qgconvert(quadruple** ∗*value*, **int** *ndigit*, **int** *trailing*, **char** ∗*buf);*

char ∗**ecvt(double** *value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign***);**

char ∗**fcvt(double** *value*, **int** *ndigit*, **int** ∗*decpt*, **int** ∗*sign***);**

char ∗**gcvt(double** *value*, **int** *ndigit*, **char** ∗*buf***);**

MT-LEVEL

MT-Safe

DESCRIPTION

econvert( ) converts the *value* to a NULL-terminated string of *ndigit* ASCII digits in *buf* and returns a pointer to *buf*. *buf* should contain at least *ndigit+1* characters. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt*. Thus *buf* == "314" and ∗*decpt* == 1 corresponds to the numerical value 3.14, while *buf* == "314" and ∗*decpt* == –1 corresponds to the numerical value .0314. If the sign of the result is negative, the word pointed to by *sign* is nonzero; otherwise it is zero. The least significant digit is rounded.

fconvert works much like **econvert**, except that the correct digit has been rounded as if for **sprintf(%w.nf)** output with *n=ndigit* digits to the right of the decimal point. *ndigit* can be negative to indicate rounding to the left of the decimal point. The return value is a pointer to *buf*. *buf* should contain at least *310+max(0,ndigit)* characters to accomodate any double-precision *value*.

gconvert( ) converts the *value* to a NULL-terminated ASCII string in *buf* and returns a pointer to *buf*. It produces *ndigit* significant digits in fixed-decimal format, like **sprintf(%w.nf)**, if possible, and otherwise in floating-decimal format, like **sprintf(%w.ne)**; in either case *buf* is ready for printing, with sign and exponent. The result corresponds to that obtained by

(void) sprintf(buf,''%w.ng'',value) ;

If *trailing*= 0, trailing zeros and a trailing point are suppressed, as in **sprintf(%g)**. If *trailing*!= 0, trailing zeros and a trailing point are retained, as in **sprintf(%#g)**.

seconvert, **sfconvert**, and **sgconvert( )** are single-precision versions of these functions, and are more efficient than the corresponding double-precision versions. A pointer rather than the value itself is passed to avoid C's usual conversion of single-precision arguments to double.

qeconvert, **qfconvert**, and **qgconvert( )** are quadruple-precision versions of these functions. **qfconvert( )** can overflow the *decimal_record* field *ds* if *value* is too large. In that case, *buf[0]* is set to zero.

ecvt( ) and **fcvt( )** are obsolete versions of **econvert( )** and **fconvert( )** that create a string in a static data area, overwritten by each call, and return values that point to that static data. These functions are therefore not reentrant.

gcvt( ) is an obsolete version of **gconvert( )** that always suppresses trailing zeros and point.

IEEE Infinities and NaNs are treated similarly by these functions. ''NaN'' is returned for NaN, and ''Inf'' or ''Infinity'' for Infinity. The longer form is produced when *ndigit* >= 8.

SEE ALSO

sprintf(3S)