

# Re: user defined function that converts string to float

---

*Source:* [http://coding.derkeiler.com/Archive/C\\_CPP/comp.lang.c/2006-02/msg00170.html](http://coding.derkeiler.com/Archive/C_CPP/comp.lang.c/2006-02/msg00170.html)

---

- *From:* "Gregory Pietsch" <GKP1@xxxxxxxx>
  - *Date:* 1 Feb 2006 18:43:28 -0800
- 

karthi wrote:

```
> hi,  
> I need user defined function that converts string to float in c.  
> since the library function atof and strtod occupies large space in  
> my processor memory I can't use it in my code.  
> regards,  
> Karthi
```

The following is a quick hack:

```
/* strtold.c - code for the atof, strtod, strtod, and strtold functions
```

```
   AUTHOR: Gregory Pietsch
```

```
   DESCRIPTION:
```

```
   The strtod, strtod, and strtold functions convert the initial  
portion of the  
   string pointed to by nptr to double, float, and long double  
representations,  
   respectively. First, they decompose the input string into three  
parts: an  
   initial, possibly empty, sequence of white-space characters (as  
specified by  
   the isspace function), a subject sequence resembling a  
floating-point  
   constant or representing infinity or NaN, and a final sequence of  
one or  
   more unrecognized characters, including the terminating null  
character of  
   the input string. Then, they attempt to convert the subject string  
to a  
   floating-point number, and return the result.
```

```
   The expected form of the subject sequence is an optional plus or  
minus sign,  
   then one of the following:
```

```
   - a nonempty sequence of decimal digits optionally containing a  
decimal-  
   point character, then an optional exponent part as defined in  
6.4.4.2;
```

```
   - a 0x or 0X, then a nonempty sequence of hexadecimal digits  
optionally  
   containing a decimal-point character, then an optional binary
```

## Re: user defined function that converts string to float

exponent

part as defined in 6.4.4.2;

- one of INF or INFINITY, ignoring case

- one of NAN or NAN(n-char-sequence\_opt), ignoring case in the NAN part,

where:

n-char-sequence:

digit

nondigit

n-char-sequence digit

n-char-sequence non-digit

The subject sequence is defined as the longest initial subsequence of the

input string, starting with the first non-white-space character, that is of

the expected form. The subject sequence contains no characters if the input

string is not of the expected form.

If the subject sequence has the expected form for a floating-point number,

the sequence of characters starting with the first digit or the decimal-

point character (whichever comes first) is interpreted as a floating

constant according to the rules of 6.4.4.2, except that the decimal-point

character is used in place of a period, and that if neither an exponent part

nor a decimal-point character appears in a decimal floating-point number,

or if a binary exponent part does not appear in a hexadecimal floating-point

number, an exponent part of the appropriate type with value zero is assumed

to follow the last digit in the string. If the subject sequence begins with

a minus sign, the sequence is interpreted as negated. (It is unspecified

whether a minus-signed sequence is converted to a negative number directly

or by negating the value resulting from converting the corresponding unsigned sequence [see F.5]; the two methods may yield different

results if

rounding is toward positive or negative infinity. In either case, the

functions honor the sign of zero if floating-point arithmetic supports

signed zeros.) A character sequence INF or INFINITY is interpreted as an

infinity, if representable in the return type, else like a floating constant

that is too large for the range of the return type. A character sequence

NAN or NAN(n-char-sequence\_opt), is interpreted as a quiet NaN, if supported

in the return type, else like a subject sequence part that does not have the

## Re: user defined function that converts string to float

expected form; the meaning of the n-char-sequences is implementation-defined (an implementation may use the n-char-sequence to determine extra information to be represented in the NaN's significand). A pointer to the final string is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

If the subject string has the hexadecimal form and FLT\_RADIX is a power of 2, the value resulting from the conversion is correctly rounded.

In other than the "C" locale, additional locale-specific subject sequences may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of nptr is stored in the object pointed to by endptr, provided that endptr is not a null pointer.

### RECOMMENDED PRACTICE:

If the subject sequence has the hexadecimal form, FLT\_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the stipulation that the error should have the correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL\_DIG (defined in <float.h>) significant digits, the result should be correctly rounded. If the subject sequence D has the decimal form and more than DECIMAL\_DIG significant digits, consider the two bounding, adjacent decimal strings L and U, both having DECIMAL\_DIG significant digits, such that the values of L, D, and U satisfy  $L \leq D \leq U$ . The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding L and U according to the current rounding direction, with the extra stipulation that the error with respect to D should have a correct sign for the current rounding direction. (DECIMAL\_DIG, defined in <float.h>, should be significantly large that L and U will usually round to the same internal floating value, but if not will round to adjacent values.)

### RETURNS:

The functions return the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the

## Re: user defined function that converts string to float

range of representable values, plus or minus HUGE\_VAL, HUGE\_VALF, or HUGE\_VALL is returned (according to the return type and sign of the value), and the value of the macro ERANGE is stored in errno. If the result underflows (7.12.1), the functions return a value whose magnitude is no greater than the smallest normalized positive number in the return type; whether errno acquires the value ERANGE is implementation-defined.

### COPYRIGHT NOTICE:

This file is placed into the public domain by the author, Gregory Pietsch.

Do with it what you wish; just don't pretend that you wrote it.  
\*/

```
/* includes needed by this file */
#include <ctype.h>
#include <errno.h>
#include <float.h>
#include <limits.h>
#include <locale.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/* defines */

/* SIG_MAX: the maximum number of significant digits we have. The
maximum
number of significant digits in a long double must be at least 35,
which is the number of significant digits in an IEEE quad precision
floating-point number (113 * log10(2.0), rounded up). */
#define SIG_MAX 40

/* _Memcasecmp: return a case-insensitive comparison of two areas of
memory. */
static int
_Memcasecmp (const void *s1, const void *s2, size_t n)
{
    const unsigned char *su1, *su2;

    for (su1 = s1, su2 = s2; 0 < n; ++su1, ++su2, --n)
    {
        if (toupper (*su1) != toupper (*su2))
            return (*su1 < *su2 ? -1 : +1);
    }
    return 0;
}

/* _Ldmul: multiply y by *px with checking */
static int
_Ldmul(long double *px, long double y)
{
    int lexp;
    long double ld;

    ld = frexpl(*px, &lexp) * y;
```

## Re: user defined function that converts string to float

```
    errno = 0;
    *px = ldexpl(ld, lexp);      /* ldexpl can overflow */
    return errno != 0 ? -1 : 0;
}

/* _Ldtento: compute x * 10^n, paranoid style */
static long double
_Ldtento(long double x, int n)
{
    long double factor, fac10;
    int errx;
    unsigned nu;

    if (n == 0 || x == 0.0L)
        return x;
    factor = 1.0L;
    if (n < 0)
    {
        /* scale down */
        nu = -(unsigned)n;
        for (fac10 = 10.0L; 0 < nu; nu >>= 1, fac10 *= fac10)
        {
            if (nu & 1)
                factor *= fac10;
        }
        errx = _Ldmul(&x, 1.0L/factor);
    }
    else
    {
        /* scale up */
        for (fac10 = 10.0L; 0 < n; n >>= 1, fac10 *= fac10)
        {
            if (n & 1)
                factor *= fac10;
        }
        errx = _Ldmul(&x, factor);
    }
    if (0 < errx)
        errno = ERANGE;
    return x;
}

/* _Stold: return the classification of the floating-point number
(FP_ZERO
if there's no number; if there's a zero, return FP_NORMAL).
The pld parameter is a possibly-returned long double;
the pnan parameter is the n-char-sequence returned if we're
returning FP_NAN. */
static int
_Stold (const char *s, char **endptr, long double *pld, char **pnan)
{
    static const char hexits[] = {
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c',
        'd', 'e',
        'f'
    };
    const char point = localeconv ()->decimal_point[0];
    const char *sc;
    char buf[SIG_MAX];
    char sign;
    long double x, fac, facpow;
    int ndigit, nsig, nzero, olead, opoint, base;
```

## Re: user defined function that converts string to float

```
size_t m;
char *p;
const char *pc;
int n;
unsigned long lo[SIG_MAX / 8 + 1], *pl;
short sexp;
long lexp;
const char *scsav, esign;

/* Get past the spaces. */
for (sc = s; isspace (*sc); ++sc)
    ;
/* Get past the initial sign. */
sign = (*sc == '-' || *sc == '+') ? *sc++ : '+';
/* Do we have INFINITY or INF? */
if (_Memcasecmp (*sc, "INFINITY", m = 8) == 0
    || _Memcasecmp (*sc, "INF", m = 3) == 0)
    {
        if (endptr)
            *endptr = sc + m;
        *pld = (sign == '-') ? -1.0L : +1.0L;
        return FP_INFINITE;
    }
/* Do we have NAN or NAN(nan-char-sequence_opt)? */
if (_Memcasecmp (*sc, "NAN(", m = 4) == 0
    || _Memcasecmp (*sc, "NAN", m = 3) == 0)
    {
        sc += m;
        if (m == 4)
            {
                /* pick off the optional n-char-sequence */
                p = strchr (sc, '(');
                if (p == 0)
                    sc--; /* pretend the '(' doesn't belong */
                else if (p == sc)
                    {
                        /* no n-char-sequence */
                        *pnan = 0;
                        sc++;
                    }
                else
                    {
                        /* This could be a little more meticulous... */
                        *pnan = malloc ((p - sc) + 1);
                        if (*pnan == 0)
                            sc--; /* out of memory */
                        else
                            {
                                memcpy (*pnan, sc, (p - sc) - 1);
                                (*pnan)[p - sc] = '\0';
                            }
                    }
            }
        else
            {
                *nan = 0;
                if (endptr)
                    *endptr = sc;
                *pld = (sign == '-') ? -1.0L : +1.0L;
                return FP_NAN;
            }
    }
/* Do we have a hexadecimal floating-point constant? */
if (_Memcasecmp (*sc, "0X", 2) == 0)
```

## Re: user defined function that converts string to float

```
{
    sc += 2;
    base = 16;
}
else
    base = 10;
/* get the digits/hexits before the exponent */
for (ndigit = nsig = nzero = 0, olead = opoint = -1;; ++sc)
{
    if (*sc == point)
    {
        /* found a decimal point */
        if (0 <= opoint)
            break;          /* already seen point */
        else
            opoint = ndigit;
    }
    else if (*sc == '0')
    {
        /* saw a zero */
        ++nzero;
        ++ndigit;
    }
    else if ((base == 16 && !isxdigit (*sc))
             || (base == 10 && !isdigit (*sc)))
        break;
    else
    {
        /* found a nonzero digit */
        if (olead < 0)
            olead = nzero;
        else
        {
            /* deliver zeros */
            for (; 0 < nzero && nsig < SIG_MAX; --nzero)
                buf[nsig++] = 0;
        }
        ++ndigit;
        if (nsig < SIG_MAX)
        {
            /* deliver digit */
            if (base == 10)
                buf[nsig++] = *sc - '0';
            else
            {
                p = strchr (hexits, tolower (*sc));
                buf[nsig++] = p - hexits;
            }
        }
    }
}
}
if (ndigit == 0)
{
    /* no digits? return not-a-long double */
    if (endpctr)
        *endpctr = (char *) s;
    return FP_ZERO;
}
/* skip trailing digits */
for (; 0 < nsig && buf[nsig - 1] == 0; --nsig)
;
/* compute significand */
```

## Re: user defined function that converts string to float

```
*pc = buf;
pl = lo + (nsig >> 3);
for (*pl = 0, n = nsig; 0 < n; --n)
{
    if ((n & 7) == 0)
        *--pl = *pc++;
    else
        *pl = *pl * base + *pc++;
}
fac = facpow = (base == 10) ? 1e8L : +4294967296.0L;
for (x = (long double) *lo, n = 0; ++n <= (nsig >> 3);)
{
    if (lo[n] != 0UL)
        x += fac * lo[n];
    fac *= facpow;
}
/* fold in any explicit exponent */
lexp = 0;
if ((base == 10 && (*sc == 'e' || *sc == 'E'))
    || (base == 16 && (*sc == 'p' || *sc == 'P')))
{
    *scsav = sc;
    esign = *++sc == '+' || *sc == '-' ? *sc++ : '+';
    if (!isdigit (*sc))
        *sc = *scsav;          /* ill-formed exponent */
    else
    {
        /* exponent looks valid */
        for (; isdigit (*sc); ++sc)
            if (lexp < 10000)
                lexp = lexp * 10 + (*sc - '0');
        /* else overflow */
        if (esign == '-')
            lexp = -lexp;
    }
}
if (endptr)
    *endptr = (char *) sc;
if (opoint < 0)
    lexp += ndigit - nsig;
else
    lexp += opoint - olead - nsig;
sexp =
    (lexp <
     SHRT_MIN) ? SHRT_MIN : ((lexp < SHRT_MAX) ? (short) lexp :
SHRT_MAX);
if (base == 10)
    *pld = _Ldtento (x, sexp);
else
    *pld = ldexpl (x, (int) (sexp) << 2);
if (sign == '-')
    *pld = -*pld;
return FP_NORMAL;
}

/* exported functions */

/* atof */
double (atof) (const char *nptr)
{
    return strtod (nptr, 0);
}
```

## Re: user defined function that converts string to float

```
/* strtod */
double (strtod) (const char *restrict nptr, char **restrict endptr)
{
    long double ld;
    char *nan_arg = 0;
    double d;

    switch (_Stold (nptr, endptr, &ld, &nan_arg))
    {
        case FP_NORMAL:
        default:
            if ((ld < 0.0L ? -ld : ld) > DBL_MAX)
            {
                errno = ERANGE;
                return (ld < 0.0L) ? -HUGE_VAL : HUGE_VAL;
            }
            else
                return (double) ld;
        case FP_ZERO:
            return 0.0;
        case FP_NAN:
            d = copysign (nan (nan_arg), (double) ld);
            if (nan_arg != 0)
                free (nan_arg);
            return d;
        case FP_INFINITE:
            return ld < 0.0L ? -HUGE_VAL : HUGE_VAL;
    }
}

/* strtodf */
float (strtodf) (const char *restrict nptr, char **restrict endptr)
{
    long double ld;
    char *nan_arg = 0;
    float f;

    switch (_Stold (nptr, endptr, &ld, &nan_arg))
    {
        case FP_NORMAL:
        default:
            if ((ld < 0.0L ? -ld : ld) > FLT_MAX)
            {
                errno = ERANGE;
                return (ld < 0.0L) ? -HUGE_VALF : HUGE_VALF;
            }
            else
                return (float) ld;
        case FP_ZERO:
            return 0.0F;
        case FP_NAN:
            f = copysignf (nanf (nan_arg), (float) ld);
            if (nan_arg != 0)
                free (nan_arg);
            return f;
        case FP_INFINITE:
            return ld < 0.0L ? -HUGE_VALF : HUGE_VALF;
    }
}

/* strtold */
```

## Re: user defined function that converts string to float

```
long double (strtold) (const char *restrict nptr, char **restrict
endpctr)
{
    long double ld;
    char *nan_arg = 0;

    switch (_Stold (nptr, endpctr, &ld, &nan_arg))
    {
        case FP_NORMAL:
        default:
            return ld;
        case FP_ZERO:
            return 0.0;
        case FP_NAN:
            ld = copysignl (nanl (nan_arg), ld);
            if (nan_arg != 0)
                free (nan_arg);
            return ld;
        case FP_INFINITE:
            return ld < 0.0L ? -HUGE_VALL : HUGE_VALL;
    }
}

/* END OF FILE */
```

Gregory Pietsch

.