



---

# **Nuclei™ C Runtime Library**

## **User Guide**

## **& Reference Manual**

## Copyright Notice

Copyright © 2018–2021 Nuclei System Technology. All rights reserved.

Nuclei™ are trademarks owned by Nuclei System Technology. All other trademarks used herein are the property of their respective owners.

The product described herein is subject to continuous development and improvement; information herein is given by Nuclei in good faith but without warranties.

This document is intended only to assist the reader in the use of the product. Nuclei System Technology shall not be liable for any loss or damage arising from the use of any information in this document, or any incorrect use of the product.

## Contact Information

Should you have any problems with the information contained herein or any suggestions, please contact Nuclei System Technology by email [support@nucleisys.com](mailto:support@nucleisys.com), or visit “Nuclei User Center” website <http://user.nucleisys.com> for supports or online discussion.

## Revision History

Rev	Revision Date	Revised Section	Revised Content
1.0.0	2021/11/25	N/A	N/A

## Table of Contents

Contact Information.....	2
Revision History.....	3
Table of Contents .....	4
Nuclei C Runtime Library User Guide & Reference Manual .....	6
Nuclei C Runtime Library for GNU Toolchain .....	6
Usage.....	6
Varieties.....	6
Quick start.....	7
exit() .....	7
Basic UART I/O functions .....	7
Thread-local storage.....	7
Heap .....	8
Runtime support.....	9
Getting to main() and then exit().....	9
Multithreaded protection for the heap.....	9
Input and output .....	9
C library API .....	27
<assert.h> .....	27
<complex.h> .....	28
<ctype.h> .....	71
<errno.h> .....	86
<fcntl.h> .....	87
<float.h> .....	94
<iso646.h> .....	96

<limits.h> .....	97
<locale.h> .....	100
<math.h> .....	103
<setjmp.h> .....	197
<stdbool.h> .....	198
<stddef.h> .....	199
<stdint.h> .....	200
<stdio.h> .....	206
<stdlib.h> .....	223
<string.h> .....	262
<time.h> .....	290
<wchar.h> .....	299
<wctype.h> .....	329
<xlocale.h> .....	349
Compiler support API .....	352
GNU library API .....	352
External function interface .....	388
I/O functions.....	388
Heap protection functions.....	390
Error and assertion functions.....	391
RTC functions .....	392
Locale functions .....	393

## Nuclei C Runtime Library User Guide & Reference Manual

Nuclei C Runtime Library is written in standard ANSI C and RISC-V assembly language and can run on RISC-V CPU. Here's a list summarising the main features of Nuclei C Runtime Library:

- Clean ISO/ANSI C source code.
- Fast assembly language floating point support.
- Conforms to standard runtime ABIs for the RISC-V architectures.

## Nuclei C Runtime Library for GNU Toolchain

### Usage

```
riscv-nuclei-elf-gcc --specs=libncrt_small.specs
```

### Varieties

#### GCC Specs

	<b>GCC Specs</b>
Fast	libncrt_fast.specs
Balanced	libncrt_balanced.specs
Small	libncrt_small.specs
Nano	libncrt_nano.specs
Pico	libncrt_pico.specs

#### Optimization

	<b>Optimization</b>
Fast	Favor speed at the expense of size
Balanced	Balanced
Small	Favor size at the expense of speed
Nano	Favor size at the expense of speed
Pico	Favor size at the expense of speed

#### Formatted I/O

	<b>int</b>	<b>long</b>	<b>long long</b>	<b>float</b>	<b>double</b>	<b>wide character</b>	<b>input character class</b>	<b>Width and precision specification</b>	<b>stdout formatting buffer (byte)</b>
--	------------	-------------	------------------	--------------	---------------	-----------------------	------------------------------	--	--

Fast	yes	yes	yes	yes	yes	yes	yes	yes	64
Balanced	yes	yes	yes	yes	yes	yes	yes	yes	64
Small	yes	yes	yes	yes	yes	yes	yes	yes	64
Nano	yes	yes	yes	no	no	no	no	yes	64
Pico	yes	no	no	no	no	no	no	no	32

### Algorithm

	Scaled-integer Algorithm
Fast	Algorithms use C-language floating-point arithmetic.
Balanced	Algorithms use C-language floating-point arithmetic.
Small	Algorithms use C-language floating-point arithmetic.
Nano	Algorithms use C-language floating-point arithmetic.
Pico	IEEE single-precision functions use scaled integer arithmetic if there is a scaled-integer implementation of the function.

## Quick start

There are four fundamental parts you need to implement for a minimal Nuclei C Runtime Library evaluation. And you can find the reference implementation in the latest **Nuclei SDK** release. The documentation for advanced features is in *External function interface*.

### exit()

```
void exit(int fd);
```

### Basic UART I/O functions

See *Using Nuclei C Runtime Library Default UART for I/O* for more details.

To simplify this tutorial, here you only need to implement 2 basic I/O functions:

```
int metal_tty_putc(int c); // UART output function
int metal_tty_getc(void);  // UART input function
```

### Thread-local storage

In the linker script, setup tdata, tbss, \_\_tls\_base and \_\_tls\_end.

```
.tdata      : ALIGN(8)
```

```
{
    PROVIDE( __tls_base = . );
    *(.tdata .tdata.* .gnu.linkonce.td.*)
} >ram AT>flash

.tbss (NOLOAD) : ALIGN(8)
{
    *(.tbss .tbss.* .gnu.linkonce.tb.*)
    *(.tcommon)
    PROVIDE( __tls_end = . );
} >ram AT>ram
```

In startup.S, load \_\_tls\_base to tp register

```
/* Initialize GP, TP and Stack Pointer SP */
.option push
.option norelax
la gp, __global_pointer$
la tp, __tls_base
.option pop
la sp, _sp
```

## Heap

In the linker script, setup \_\_heap\_start and \_\_heap\_end. You need to align the heap to a 16-byte boundary and reserve \_\_HEAP\_SIZE bytes for it.

```
.heap (NOLOAD) :
{
    . = ALIGN(16);
    PROVIDE( __heap_start = . );
    . += __HEAP_SIZE;
    . = ALIGN(16);
    PROVIDE( __heap_limit = . );
} >ram AT>ram

.stack ORIGIN(ram) + LENGTH(ram) - __STACK_SIZE (NOLOAD) :
{
    . = ALIGN(16);
    PROVIDE( _heap_end = . );
    PROVIDE( __heap_end = . );
    PROVIDE( __StackLimit = . );
    PROVIDE( __StackBottom = . );
    . += __STACK_SIZE;
```



```
. = ALIGN(16);  
PROVIDE( __StackTop = . );  
PROVIDE( _sp = . );  
} >ram AT>ram  
}
```

## Runtime support

This section describes how to set up the execution environment for the C library.

### Getting to main() and then exit()

Before entering main() the execution environment must be set up such that the C standard library will function correctly.

This section does not describe the compiler or linker support for placing code and data into memory, how to configure any RAM, or how to zero memory required for zero-initialized data. For this, please refer to your toolset compiler and linker documentation.

Nor does this section document how to call constructors and destructors in the correct order. Again, refer to your toolset manuals.

### At-exit function support

After returning from main() or by calling exit(), any registered atexit functions must be called to close down. To do this, call \_\_SEGGER\_RTL\_execute\_at\_exit\_fns() from the runtime startup immediately after the call to main().

### Multithreaded protection for the heap

Heap functions (allocation, reallocation, deallocation) can be protected from reentrancy in a multithreaded environment by implementing lock and unlock functions. By default, these functions do nothing and memory allocation functions are not protected.

See [\\_\\_SEGGER\\_RTL\\_X\\_heap\\_lock](#) and [\\_\\_SEGGER\\_RTL\\_X\\_heap\\_unlock](#).

### Input and output

The way characters and strings are printed and scanned can be configured in multiple ways. This section describes how a generic implementation works, how to optimize input and output for other

technologies such as SEGGER RTT and SEGGER semihosting, and how to optimized for UART-style I/O.

## Standard input and output

Standard input and output are performed using the low-level functions

`__SEGGER_RTL_X_file_write()` and `__SEGGER_RTL_X_file_read()`, These functions are defined in the file `__SEGGER_RTL.h` as follows:

```
int __SEGGER_RTL_X_file_read (__SEGGER_RTL_FILE *stream, char *s, unsigned len);
int __SEGGER_RTL_X_file_write (__SEGGER_RTL_FILE *stream, const char *s, unsigned len);
```

The type `__SEGGER_RTL_FILE` and its corresponding standard C version `FILE` are defined opaquely by `__SEGGER_RTL.h` as:

```
typedef struct __SEGGER_RTL_FILE_IMPL __SEGGER_RTL_FILE;
typedef struct __SEGGER_RTL_FILE_IMPL FILE;
```

This leaves the exact structure of a `FILE` and the implementation of file I/O to the library integrator. The following are sample implementations for SEGGER RTT, SEGGER Semihosting, and a version that supports only output to a UART.

## Using Nuclei C Runtime Library Default UART for I/O

### Complete listing

```

/*****
*                               (c) SEGGER Microcontroller GmbH                               *
*                               The Embedded Experts                                       *
*                               www.segger.com                                             *
*****-----
----- END-OF-HEADER -----

*/

/*****
*                               #include section                                       *
*****

*/

#include "__SEGGER_RTL_Int.h"
#include "stdio.h"

/*****/pre>

```

```
*      Local types
*****

*/

struct __SEGGER_RTL_FILE_impl {
    int handle; // At least one field required (but unused) to ensure
                // the three file descriptors have unique addresses.
};

/*****

*      Prototypes
*****

*/

int metal_tty_putc(int c); // UART output function
int metal_tty_getc(void);  // UART input function

/*****

*      Static data
*****

*/

static FILE __SEGGER_RTL_stdin  = { 0 };
static FILE __SEGGER_RTL_stdout = { 1 };
static FILE __SEGGER_RTL_stderr = { 2 };

/*****

*      Public data
*****

*/

FILE __SEGGER_RTL_PUBLIC_API *stdin  = &__SEGGER_RTL_stdin;
FILE __SEGGER_RTL_PUBLIC_API *stdout = &__SEGGER_RTL_stdout;
```

---

```
FILE __SEGGER_RTL_PUBLIC_API *stderr = &__SEGGER_RTL_stderr;

/*****

*      Public code

*****/

*/

/*****

*      __SEGGER_RTL_X_file_read()

*      Function description

*      Read data from file.

*      Parameters

*      stream - Pointer to file to read from.

*      s      - Pointer to object that receives the input.

*      len    - Number of characters to read from file.

*      Return value

*      >= 0 - Success.

*      < 0 - Failure.

*/

int __SEGGER_RTL_PUBLIC_API __SEGGER_RTL_X_file_read(__SEGGER_RTL_FILE * stream,
char * s, unsigned len)
{
    int r;
    //
    if (stream == stdin) {
        r = 0;
        while (len > 0) {
            *s++ = metal_tty_getc();
            --len;
        }
    }
}
```

---

```
    }
} else {
    return EOF;
}
//
return r;
}

/*****
*    __SEGGER_RTL_X_file_write()
*    Function description
*    Write data to file.
*    Parameters
*    stream - Pointer to file to write to.
*    s      - Pointer to object to write to file.
*    len    - Number of characters to write to the file.
*    Return value
*    >= 0 - Success.
*    < 0 - Failure.
*    Additional information
*    Writing to any file other than stdout or stderr results in an error.
*/
int __SEGGER_RTL_PUBLIC_API __SEGGER_RTL_X_file_write(__SEGGER_RTL_FILE *stream,
const char *s, unsigned len) {
    int r;
    //
    if (stream == stdout || stream == stderr) {
        while (len > 0) {
```

```
metal_tty_putc(*s++);
--len;
}
r = 0;
} else {
r = EOF;
}
//
return r;
}

/*****
*    __SEGGER_RTL_X_file_ungetc()
*    Function description
*    Push character back to stream.
*    Parameters
*    stream - Pointer to file to push back to.
*    c      - Character to push back.
*    Return value
*    >= 0 - Success.
*    < 0 - Failure.
*    Additional information
*    As input from the UART is not supported, this function always fails.
*/
int __SEGGER_RTL_PUBLIC_API __SEGGER_RTL_X_file_ungetc(__SEGGER_RTL_FILE *stream, int
c) {
return EOF;
}
```

```

/***** End of file *****/

```

## Using SEGGER RTT for I/O

### Complete listing

```

/*****
*
*      (c) SEGGER Microcontroller GmbH
*      The Embedded Experts
*      www.segger.com
*
*****/

----- END-OF-HEADER -----
*/

/*****
*
*      #include section
*
*****/

#include "__SEGGER_RTL_Int.h"
#include "stdio.h"
#include "RTT/SEGGER_RTT.h"

/*****
*
*      Local types
*
*****/

struct __SEGGER_RTL_FILE_impl {
    int handle;
};

/*****
*
*      Static data
*
*****/

static FILE __SEGGER_RTL_stdin_file = { 0 }; // stdin reads from RTT buffer #0
static FILE __SEGGER_RTL_stdout_file = { 0 }; // stdout writes to RTT buffer #0

```

```
static FILE __SEGGER_RTL_stderr_file = { 0 }; // stdout writes to RTT buffer #0
static int __SEGGER_RTL_stdin_ungot = EOF;

/*****
 *
 *      Public data
 *
 *****/

FILE *stdin  = &__SEGGER_RTL_stdin_file;
FILE *stdout = &__SEGGER_RTL_stdout_file;
FILE *stderr = &__SEGGER_RTL_stderr_file;

/*****
 *
 *      Static code
 *
 *****/

/*****
 *
 *      __SEGGER_RTL_stdin_getc()
 *
 *      Function description
 *      Get character from standard input.
 *
 *      Return value
 *      Character received.
 *
 *      Additional information
 *      This function never fails to deliver a character.
 */
static char __SEGGER_RTL_stdin_getc(void) {
    int  r;
    char c;
    //
    if (__SEGGER_RTL_stdin_ungot != EOF) {
        c = __SEGGER_RTL_stdin_ungot;
        __SEGGER_RTL_stdin_ungot = EOF;
    } else {
        do {
            r = SEGGER_RTT_Read(stdin->handle, &c, 1);
        } while (r == 0);
    }
}
```



```

//
return c;
}

/*****
*
*      Public code
*
*****/

/*****
*
*      __SEGGER_RTL_X_file_read()
*
*      Function description
*      Read data from file.
*
*      Parameters
*      stream - Pointer to file to read from.
*      s      - Pointer to object that receives the input.
*      len    - Number of characters to read from file.
*
*      Return value
*      >= 0 - Success.
*      < 0 - Failure.
*
*      Additional information
*      Reading from any stream other than stdin results in an error.
*/
int __SEGGER_RTL_X_file_read(__SEGGER_RTL_FILE * stream,
                             char * s,
                             unsigned len) {
    int c;
    //
    if (stream == stdin) {
        c = 0;
        while (len > 0) {
            *s++ = __SEGGER_RTL_stdin_getc();
            --len;
        }
    } else {
        c = EOF;
    }
    //
    return c;
}

```

```
}

/*****
*
*   __SEGGER_RTL_X_file_write()
*
*   Function description
*   Write data to file.
*
*   Parameters
*   stream - Pointer to file to write to.
*   s      - Pointer to object to write to file.
*   len    - Number of characters to write to the file.
*
*   Return value
*   >= 0 - Success.
*   < 0 - Failure.
*
*   Additional information
*   stdout is directed to RTT buffer #0; stderr is directed to RTT buffer #1;
*   writing to any stream other than stdout or stderr results in an error
*/
int __SEGGER_RTL_X_file_write(__SEGGER_RTL_FILE *stream, const char *s, unsigned len) {
    return SEGGER_RTT_Write(stream->handle, s, len);
}

/*****
*
*   __SEGGER_RTL_X_file_unget()
*
*   Function description
*   Push character back to stream.
*
*   Parameters
*   stream - Pointer to file to push back to.
*   c      - Character to push back.
*
*   Return value
*   >= 0 - Success.
*   < 0 - Failure.
*
*   Additional information
*   Push-back is only supported for standard input, and
*   only a single-character pushback buffer is implemented.
*/
```

```
int __SEGGER_RTL_X_file_ungetc(__SEGGER_RTL_FILE *stream, int c) {
    if (stream == stdin) {
        if (c != EOF && __SEGGER_RTL_stdin_ungot == EOF) {
            __SEGGER_RTL_stdin_ungot = c;
        } else {
            c = EOF;
        }
    } else {
        c = EOF;
    }
    //
    return c;
}

/***** End of file *****/
```

## Using SEGGER semihosting for I/O

### Complete listing

```

/*****
*
*          (c) SEGGER Microcontroller GmbH
*          The Embedded Experts
*          www.segger.com
*
*****/

----- END-OF-HEADER -----
*/

/*****
*
*      #include section
*
*****/

#include "__SEGGER_RTL_Int.h"
#include "stdio.h"
#include "SEMIHOST/SEGGER_SEMIHOST.h"

/*****
*
*      Local types
*
*****/

```

```

struct __SEGGER_RTL_FILE_impl {
    int handle;
};

/*****
*
*      Static data
*
*****/

static FILE __SEGGER_RTL_stdin_file  = { SEGGER_SEMIHOST_STDIN  };
static FILE __SEGGER_RTL_stdout_file = { SEGGER_SEMIHOST_STDOUT };
static FILE __SEGGER_RTL_stderr_file = { SEGGER_SEMIHOST_ERROUT };
static int  __SEGGER_RTL_stdin_ungot = EOF;

/*****
*
*      Public data
*
*****/

FILE *stdin  = &__SEGGER_RTL_stdin_file;
FILE *stdout = &__SEGGER_RTL_stdout_file;
FILE *stderr = &__SEGGER_RTL_stderr_file;

/*****
*
*      Static code
*
*****/

/*****
*
*      __SEGGER_RTL_stdin_getc()
*
*      Function description
*      Get character from standard input.
*
*      Return value
*      >= 0   - Character read.
*      == EOF - End of stream or error reading.
*****/

```

```

*
* Additional information
* This function never fails to deliver a character.
*/
static int __SEGGER_RTL_stdin_getc(void) {
    int r;
    char c;
    //
    if (__SEGGER_RTL_stdin_ungot != EOF) {
        c = __SEGGER_RTL_stdin_ungot;
        __SEGGER_RTL_stdin_ungot = EOF;
        r = 0;
    } else {
        r = SEGGER_SEMIHOST_ReadC();
    }
    //
    return r < 0 ? EOF : c;
}

/*****
*
* Public code
*
*****/

/*****
*
* __SEGGER_RTL_X_file_read()
*
* Function description
* Read data from file.
*
* Parameters
* stream - Pointer to file to read from.
* s       - Pointer to object that receives the input.
* len     - Number of characters to read from file.
*
* Return value
* >= 0 - Success.
* < 0 - Failure.
*
* Additional information
* Reading from any stream other than stdin results in an error.
*/
int __SEGGER_RTL_X_file_read(__SEGGER_RTL_FILE * stream,

```

```

char * s,
unsigned len) {

int c;
//
if (stream == stdin) {
    c = 0;
    while (len > 0) {
        *s++ = __SEGGER_RTL_stdin_getc();
        --len;
    }
} else {
    c = SEGGER_SEMIHOST_Read(stream->handle, s, len);
}
//
return c;
}

/*****
*
*    __SEGGER_RTL_X_file_write()
*
*  Function description
*    Write data to file.
*
*  Parameters
*    stream - Pointer to file to write to.
*    s       - Pointer to object to write to file.
*    len     - Number of characters to write to the file.
*
*  Return value
*    >= 0 - Success.
*    < 0 - Failure.
*/
int __SEGGER_RTL_X_file_write(__SEGGER_RTL_FILE *stream, const char *s, unsigned len) {
    int r;
    //
    r = SEGGER_SEMIHOST_Write(stream->handle, s, len);
    if (r < 0) {
        r = EOF;
    }
    //
    return r;
}

/*****

```

```

*
*   __SEGGER_RTL_X_file_unget()
*
*   Function description
*   Push character back to stream.
*
*   Parameters
*   stream - Pointer to stream to push back to.
*   c      - Character to push back.
*
*   Return value
*   >= 0 - Success.
*   < 0 - Failure.
*
*   Additional information
*   Push-back is only supported for standard input, and
*   only a single-character pushback buffer is implemented.
*/
int __SEGGER_RTL_X_file_unget(__SEGGER_RTL_FILE *stream, int c) {
    if (stream == stdin) {
        if (c != EOF && __SEGGER_RTL_stdin_ungot == EOF) {
            __SEGGER_RTL_stdin_ungot = c;
        } else {
            c = EOF;
        }
    } else {
        c = EOF;
    }
    //
    return c;
}

/***** End of file *****/

```

## Using a UART for I/O

### Complete listing

```

/*****
*
*   (c) SEGGER Microcontroller GmbH
*   The Embedded Experts
*   www.segger.com
*
*****/

----- END-OF-HEADER -----
*/

```

```
/*
 *
 *      #include section
 *
 */

#include "__SEGGER_RTL_Int.h"
#include "stdio.h"

/*
 *
 *      Local types
 *
 */

struct __SEGGER_RTL_FILE_impl {
    int handle; // At least one field required (but unused) to ensure
                // the three file descriptors have unique addresses.
};

/*
 *
 *      Prototypes
 *
 */

int metal_tty_putc(int c); // UART output function

/*
 *
 *      Static data
 *
 */

static FILE __SEGGER_RTL_stdin = { 0 };
static FILE __SEGGER_RTL_stdout = { 1 };
static FILE __SEGGER_RTL_stderr = { 2 };

/*
 *
```



```

*      Public data
*
*****
*/

FILE *stdin  = &__SEGGER_RTL_stdin;
FILE *stdout = &__SEGGER_RTL_stdout;
FILE *stderr = &__SEGGER_RTL_stderr;

/*****
*
*      Public code
*
*****
*/

/*****
*
*      __SEGGER_RTL_X_file_read()
*
*      Function description
*      Read data from file.
*
*      Parameters
*      stream - Pointer to file to read from.
*      s      - Pointer to object that receives the input.
*      len    - Number of characters to read from file.
*
*      Return value
*      >= 0 - Success.
*      < 0 - Failure.
*
*      Additional information
*      As input from the UART is not supported, this function always fails.
*/
int __SEGGER_RTL_X_file_read(__SEGGER_RTL_FILE * stream,
                             char               * s,
                             unsigned          len) {
    return EOF;
}

/*****
*
*      __SEGGER_RTL_X_file_write()
*
*      Function description

```

---

```
*   Write data to file.
*
* Parameters
*   stream - Pointer to file to write to.
*   s      - Pointer to object to write to file.
*   len    - Number of characters to write to the file.
*
* Return value
*   >= 0 - Success.
*   < 0 - Failure.
*
* Additional information
*   Writing to any file other than stdout or stderr results in an error.
*/
int __SEGGER_RTL_X_file_write(__SEGGER_RTL_FILE *stream, const char *s, unsigned len) {
    int r;
    //
    if (stream == stdout || stream == stderr) {
        while (len > 0) {
            metal_tty_putc(*s++);
            --len;
        }
        r = 0;
    } else {
        r = EOF;
    }
    //
    return r;
}

/*****
*
*   __SEGGER_RTL_X_file_ungetc()
*
* Function description
*   Push character back to stream.
*
* Parameters
*   stream - Pointer to file to push back to.
*   c      - Character to push back.
*
* Return value
*   >= 0 - Success.
*   < 0 - Failure.
*
*****/
```

---

```
* Additional information
* As input from the UART is not supported, this function always fails.
*/
int __SEGGER_RTL_X_file_unget(__SEGGER_RTL_FILE *stream, int c) {
    return EOF;
}

/***** End of file *****/
```

## C library API

### <assert.h>

#### Assertion functions

Function	Description
assert	Place assertion.

#### *assert*

##### Description

Place assertion.

##### Definition

```
#define assert(e)    ...
```

##### Additional information

If NDEBUG is defined as a macro name at the point in the source file where <assert.h> is included, the assert() macro is defined as:

```
#define assert(ignore) ((void)0)
```

If NDEBUG is not defined as a macro name at the point in the source file where <assert.h> is included, the assert() macro expands to a void expression that calls \_\_SEGGER\_RTL\_X\_assert().

When such an assert is executed and e is false, assert() calls the function \_\_SEGGER\_RTL\_X\_assert() with information about the particular call that failed: the text of the argument, the name of the source file, and the source line number. These are the stringized expression and the values of the preprocessing macros \_\_FILE\_\_ and \_\_LINE\_\_.

##### Notes

The assert() macro is redefined according to the current state of NDEBUG each time that <assert.h> is included.

## <complex.h>

Nuclei C Runtime Library provides complex math library functions, including all of those required by ISO C99. These functions are implemented to balance performance with correctness. Because producing the correctly rounded result may be prohibitively expensive, these functions are designed to efficiently produce a close approximation to the correctly rounded result. In most cases, the result produced is within +/-1 ulp of the correctly rounded result, though there may be cases where there is greater inaccuracy.

### Manipulation functions

Function	Description
<code>cabs()</code>	Compute magnitude, double complex.
<code>cabsf()</code>	Compute magnitude, float complex.
<code>cabsl()</code>	Compute magnitude, long double complex.
<code>carg()</code>	Compute phase, double complex.
<code>cargf()</code>	Compute phase, float complex.
<code>cargl()</code>	Compute phase, long double complex.
<code>cimag()</code>	Imaginary part, double complex.
<code>cimagf()</code>	Imaginary part, float complex.
<code>cimagl()</code>	Imaginary part, long double complex.
<code>creal()</code>	Real part, double complex.
<code>crealf()</code>	Real part, float complex.
<code>creall()</code>	Real part, long double complex.
<code>cproj()</code>	Project, double complex.
<code>cprojf()</code>	Project, float complex.
<code>cprojl()</code>	Project, long double complex.
<code>conj()</code>	Conjugate, double complex.
<code>conjf()</code>	Conjugate, float complex.
<code>conjl()</code>	Conjugate, long double complex.

### `cabs()`

#### Description

Compute magnitude, double complex.

#### Prototype

```
double cabs(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute magnitude of.

## Return value

The magnitude of x,  $|x|$ .

## *cabsf()*

## Description

Compute magnitude, float complex.

## Prototype

```
float cabsf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute magnitude of.

## Return value

The magnitude of x,  $|x|$ .

## *cabs()*

## Description

Compute magnitude, long double complex.

## Prototype

```
long double cabsl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute magnitude of.

## Return value

The magnitude of x,  $|x|$ .

### *carg()*

#### Description

Compute phase, double complex.

#### Prototype

```
double carg(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Value to compute phase of.

#### Return value

The phase of x.

### *cargf()*

#### Description

Compute phase, float complex.

#### Prototype

```
float cargf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Value to compute phase of.

#### Return value

The phase of x.

### *cargl()*

#### Description

Compute phase, long double complex.

#### Prototype

```
long double cargl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

#### Parameters

---

Parameter	Description
x	Value to compute phase of.

Return value

The phase of x.

*cimag()*

Description

Imaginary part, double complex.

Prototype

```
double cimag(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

---

Parameter	Description
x	Argument.

Return value

The imaginary part of the complex value.

*cimagf()*

Description

Imaginary part, float complex.

Prototype

```
float cimagf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

---

Parameter	Description
x	Argument.

Return value

The imaginary part of the complex value.

*cimagl()*

Description

---

Imaginary part, long double complex.

Prototype

```
long double cimagl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The imaginary part of the complex value.

[\*creal\(\)\*](#)

Description

Real part, double complex.

Prototype

```
double creal(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

The real part of the complex value.

[\*crealf\(\)\*](#)

Description

Real part, float complex.

Prototype

```
float crealf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.



## Return value

The real part of the complex value.

## *creall()*

## Description

Real part, long double complex.

## Prototype

```
long double creall(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Argument.

## Return value

The real part of the complex value.

## *cproj()*

## Description

Project, double complex.

## Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cproj(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to project.

## Return value

The projection of x to the Reimann sphere.

## Additional information

x projects to x, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If x has an infinite part, then cproj(x) is be equivalent to:

- $\text{INFINITY} + I * \text{copysign}(0.0, \text{cimag}(x))$

## *cprojf()*

### Description

Project, float complex.

### Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cprojf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Value to project.

### Return value

The projection of x to the Reimann sphere.

### Additional information

x projects to x, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If x has an infinite part, then `cproj(x)` is be equivalent to:

- `INFINITY + I * copysign(0.0, cimag(x))`

## *cprojl()*

### Description

Project, long double complex.

### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cprojl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Value to project.

### Return value

The projection of x to the Reimann sphere.

### Additional information

`x` projects to `x`, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If `x` has an infinite part, then `cproj(x)` is be equivalent to:

- `INFINITY + I * copysignl(0.0, cimagl(x))`

### `conj()`

Description

Conjugate, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX conj(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
<code>x</code>	Value to conjugate.

Return value

The complex conjugate of `x`.

### `conjf()`

Description

Conjugate, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX conjf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
<code>x</code>	Value to conjugate.

Return value

The complex conjugate of `x`.

### `conjl()`

Description

Conjugate, long double complex.

## Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX conjl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to conjugate.

## Return value

The complex conjugate of x.

## Trigonometric functions

Function	Description
<a href="#">csin()</a>	Compute sine, double complex.
<a href="#">csinf()</a>	Compute sine, float complex.
<a href="#">csinl()</a>	Compute sine, long double complex.
<a href="#">ccos()</a>	Compute cosine, double complex.
<a href="#">ccosf()</a>	Compute cosine, float complex.
<a href="#">ccosl()</a>	Compute cosine, long double complex.
<a href="#">ctan()</a>	Compute tangent, double complex.
<a href="#">ctanf()</a>	Compute tangent, float complex.
<a href="#">ctanl()</a>	Compute tangent, long double complex.
<a href="#">casin()</a>	Compute inverse sine, double complex.
<a href="#">casinf()</a>	Compute inverse sine, float complex.
<a href="#">casinl()</a>	Compute inverse sine, long double complex.
<a href="#">cacos()</a>	Compute inverse cosine, double complex.
<a href="#">cacosf()</a>	Compute inverse cosine, float complex.
<a href="#">cacosl()</a>	Compute inverse cosine, long double complex.
<a href="#">catan()</a>	Compute inverse tangent, double complex.
<a href="#">catanf()</a>	Compute inverse tangent, float complex.
<a href="#">catanl()</a>	Compute inverse tangent, long double complex.

## [csin\(\)](#)

## Description

Compute sine, double complex.

## Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX csin(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute sine of.

## Return value

The sine of x.

*csinf()*

## Description

Compute sine, float complex.

## Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX csinf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute sine of.

## Return value

The sine of x.

*csinl()*

## Description

Compute sine, long double complex.

## Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX csinl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute sine of.

## Return value

The sine of x.

*ccos()*

Description

Compute cosine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ccos(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute cosine of.

Return value

The cosine of x.

*ccosf()*

Description

Compute cosine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ccosf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute cosine of.

Return value

The cosine of x.

*ccosl()*

Description

Compute cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ccosl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute cosine of.

## Return value

The cosine of x.

## *ctan()*

## Description

Compute tangent, double complex.

## Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ctan(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute tangent of.

## Return value

The tangent of x.

## *ctanf()*

## Description

Compute tangent, float complex.

## Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ctanf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute tangent of.

## Return value

The tangent of x.

## *ctanl()*

### Description

Compute tangent, long double complex.

### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ctanl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Value to compute tangent of.

### Return value

The tangent of x.

## *casin()*

### Description

Compute inverse sine, double complex.

### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX casin(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Argument.

### Return value

Inverse sine of x.

### Notes

$\text{casin}(z) = -i \text{casinh}(i.z)$

## *casinf()*

### Description

Compute inverse sine, float complex.

### Prototype



---

```
__SEGGER_RTL_FLOAT32_C_COMPLEX casinf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Argument.

#### Return value

Inverse sine of x.

#### Notes

$\text{casin}(z) = -i \text{casinh}(i.z)$

[\*casinl\(\)\*](#)

#### Description

Compute inverse sine, long double complex.

#### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX casinl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Argument.

#### Return value

Inverse sine of x.

#### Notes

$\text{casinl}(z) = -i \text{casinhl}(i.z)$

[\*cacos\(\)\*](#)

#### Description

Compute inverse cosine, double complex.

#### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cacos(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

#### Parameters

---

---

Parameter	Description
x	Value to compute inverse cosine of.

Return value

The inverse cosine of x.

*ccosf()*

Description

Compute inverse cosine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ccosf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

---

Parameter	Description
x	Value to compute inverse cosine of.

Return value

The inverse cosine of x.

*ccosl()*

Description

Compute inverse cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ccosl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

---

Parameter	Description
x	Value to compute inverse cosine of.

Return value

The inverse cosine of x.

*ccatan()*

Description

Compute inverse tangent, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX catan(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse tangent of x.

Notes

$\text{catan}(z) = -i \text{catanh}(i.z)$

[catanf\(\)](#)

Description

Compute inverse tangent, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX catanf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Inverse tangent of x.

Notes

$\text{catan}(z) = -i \text{catanh}(i.z)$

[catanl\(\)](#)

Description

Compute inverse tangent, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX catanl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Argument.

#### Return value

Inverse tangent of x.

#### Notes

$\text{catanl}(z) = -i \text{catanhl}(i.z)$

### Hyperbolic functions

Function	Description
<a href="#">csinh()</a>	Compute hyperbolic sine, double complex.
<a href="#">csinhf()</a>	Compute hyperbolic sine, float complex.
<a href="#">csinhl()</a>	Compute hyperbolic sine, long double complex.
<a href="#">ccosh()</a>	Compute hyperbolic cosine, double complex.
<a href="#">ccoshf()</a>	Compute hyperbolic cosine, float complex.
<a href="#">ccoshl()</a>	Compute hyperbolic cosine, long double complex.
<a href="#">ctanh()</a>	Compute hyperbolic tangent, double complex.
<a href="#">ctanhf()</a>	Compute hyperbolic tangent, float complex.
<a href="#">ctanhl()</a>	Compute hyperbolic tangent, long double complex.
<a href="#">casinh()</a>	Compute inverse hyperbolic sine, double complex.
<a href="#">casinhf()</a>	Compute inverse hyperbolic sine, float complex.
<a href="#">casinhl()</a>	Compute inverse hyperbolic sine, long double complex.
<a href="#">cacosh()</a>	Compute inverse hyperbolic cosine, double complex.
<a href="#">cacoshf()</a>	Compute inverse hyperbolic cosine, float complex.
<a href="#">cacoshl()</a>	Compute inverse hyperbolic cosine, long double complex.
<a href="#">catanh()</a>	Compute inverse hyperbolic tangent, double complex.
<a href="#">catanhf()</a>	Compute inverse hyperbolic tangent, float complex.
<a href="#">catanhl()</a>	Compute inverse hyperbolic tangent, long double complex.

#### [csinh\(\)](#)

#### Description

Compute hyperbolic sine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX csinh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

The hyperbolic sine of x according to the following table:

Argument	csinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	±0 + NaNi, sign of real part unspecified
+0 + NaNi	±0 + NaNi, sign of real part unspecified
a + ∞i	NaN + NaNi, for positive finite a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + +∞×sin(b).i for positive finite b
+∞ + ∞i	±∞ + NaNi, sign of real part unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{csinh}(\text{conj}(z)) = \text{conj}(\text{csinh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{csinh}(-z) = -\text{csinh}(z)$ .

*csinhf()*

Description

Compute hyperbolic sine, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX csinhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

## Return value

The hyperbolic sine of x according to the following table:

Argument	csinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	±0 + NaNi, sign of real part unspecified
+0 + NaNi	±0 + NaNi, sign of real part unspecified
a + ∞i	NaN + NaNi, for positive finite a
a + NaNi	NaN + NaNi, for finite nonzero a
+∞ + 0i	+∞ + 0i
+∞ + bi	+∞×cos(b) + +∞×sin(b).i for positive finite b
+∞ + ∞i	±∞ + NaNi, sign of real part unspecified
+∞ + NaNi	±∞ + NaNi, sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{csinh}(\text{conj}(z)) = \text{conj}(\text{csinh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{csinh}(-z) = -\text{csinh}(z)$ .

## *csinhl()*

## Description

Compute hyperbolic sine, long double complex.

## Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX csinh1(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

Return value

The hyperbolic sine of x according to the following table:

Argument	$\cosh(\text{Argument})$
+0 + 0i	+0 + 0i
+0 + $\infty$ i	$\pm 0 + \text{NaNi}$ , sign of real part unspecified
+0 + NaNi	$\pm 0 + \text{NaNi}$ , sign of real part unspecified
a + $\infty$ i	NaN + NaNi, for positive finite a
a + NaNi	NaN + NaNi, for finite nonzero a
$+\infty$ + 0i	$+\infty$ + 0i
$+\infty$ + bi	$+\infty \times \cos(b) + +\infty \times \sin(b).i$ for positive finite b
$+\infty$ + $\infty$ i	$\pm \infty + \text{NaNi}$ , sign of real part unspecified
$+\infty$ + NaNi	$\pm \infty + \text{NaNi}$ , sign of real part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\cosh(\text{conj}(z)) = \text{conj}(\cosh(z))$ .

For arguments with a negative real component, use the equality:

- $\cosh(-z) = -\cosh(z)$ .

*ccosh()*

Description

Compute hyperbolic cosine, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ccosh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

## Return value

The hyperbolic cosine of  $x$  according to the following table:

Argument	$\cosh(\text{Argument})$
$+0 + 0i$	$+1 + 0i$
$+0 + \infty i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$+0 + \text{NaN}i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$a + \infty i$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $a$
$+\infty + 0i$	$+\infty + 0i$
$+\infty + bi$	$+\infty \times \cos(b) + \text{Inf} \times \sin(b).i$ for finite nonzero $b$
$+\infty + \infty i$	$+\infty + \text{NaN}i$
$+\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + 0i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for all nonzero $b$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\cosh(\text{conj}(z)) = \text{conj}(\cosh(z))$ .

## *ccoshf()*

### Description

Compute hyperbolic cosine, float complex.

### Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ccoshf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

### Parameters

Parameter	Description
$x$	Value to compute hyperbolic cosine of.

## Return value

The hyperbolic cosine of  $x$  according to the following table:

Argument	$\cosh(\text{Argument})$
$+0 + 0i$	$+1 + 0i$



$+0 + \infty i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$+0 + \text{NaN}i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$a + \infty i$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $a$
$+\infty + 0i$	$+\infty + 0i$
$+\infty + bi$	$+\infty \times \cos(b) + \text{Inf} \times \sin(b).i$ for finite nonzero $b$
$+\infty + \infty i$	$+\infty + \text{NaN}i$
$+\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + 0i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for all nonzero $b$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{ccosh}(\text{conj}(z)) = \text{conj}(\text{ccosh}(z))$ .

### *ccoshl()*

#### Description

Compute hyperbolic cosine, long double complex.

#### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ccoshl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

#### Parameters

Parameter	Description
$x$	Value to compute hyperbolic cosine of.

#### Return value

The hyperbolic cosine of  $x$  according to the following table:

Argument	$\text{ccosh}(\text{Argument})$
$+0 + 0i$	$+1 + 0i$
$+0 + \infty i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$+0 + \text{NaN}i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$a + \infty i$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $a$

$+\infty + 0i$	$+\infty + 0i$
$+\infty + bi$	$+\infty \times \cos(b) + \text{Inf} \times \sin(b).i$ for finite nonzero $b$
$+\infty + \infty i$	$+\infty + \text{NaN}i$
$+\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + 0i$	$\text{NaN} + \pm 0i$ , sign of imaginary part unspecified
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for all nonzero $b$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{ccosh}(\text{conj}(z)) = \text{conj}(\text{ccosh}(z))$ .

### *ctanh()*

#### Description

Compute hyperbolic tangent, double complex.

#### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX ctanh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

#### Parameters

Parameter	Description
$x$	Value to compute hyperbolic tangent of.

#### Return value

The hyperbolic tangent of  $x$  according to the following table:

Argument	$\text{ctanh}(\text{Argument})$
$+0 + 0i$	$+0 + 0i$
$a + \infty i$	$\text{NaN} + \text{NaN}i$ , for finite $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite $a$
$+\infty + bi$	$+1 + \sin(2b) \times 0i$ for positive-signed finite $b$
$+\infty + \infty i$	$+1 + \pm 0i$ , sign of imaginary part unspecified
$+\infty + \text{NaN}i$	$+1 + \pm 0i$ , sign of imaginary part unspecified
$\text{NaN} + 0i$	$\text{NaN} + 0i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for all nonzero $b$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\operatorname{ctanh}(\operatorname{conj}(z)) = \operatorname{conj}(\operatorname{ctanh}(z))$ .

For arguments with a negative real component, use the equality:

- $\operatorname{ctanh}(-z) = -\operatorname{ctanh}(z)$ .

### *ctanhf()*

#### Description

Compute hyperbolic tangent, float complex.

#### Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX ctanhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

#### Return value

The hyperbolic tangent of x according to the following table:

Argument	$\operatorname{ctanh}(\text{Argument})$
+0 + 0i	+0 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + bi	+1 + sin(2b)×0i for positive-signed finite b
+∞ + ∞i	+1 + ±0i, sign of imaginary part unspecified
+∞ + NaNi	+1 + ±0i, sign of imaginary part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\operatorname{ctanhf}(\operatorname{conj}(z)) = \operatorname{conj}(\operatorname{ctanhf}(z))$ .

For arguments with a negative real component, use the equality:

- $\operatorname{ctanhf}(-z) = -\operatorname{ctanhf}(z)$ .

## *ctanh()*

### Description

Compute hyperbolic tangent, long double complex.

### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX ctanh1(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

### Return value

The hyperbolic tangent of x according to the following table:

Argument	ctanh(Argument)
+0 + 0i	+0 + 0i
a + ∞i	NaN + NaNi, for finite a
a + NaNi	NaN + NaNi, for finite a
+∞ + bi	+1 + sin(2b)×0i for positive-signed finite b
+∞ + ∞i	+1 + ±0i, sign of imaginary part unspecified
+∞ + NaNi	+1 + ±0i, sign of imaginary part unspecified
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for all nonzero b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{ctanh}(\text{conj}(z)) = \text{conj}(\text{ctanh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{ctanh}(-z) = -\text{ctanh}(z)$ .

## *casinh()*

### Description

Compute inverse hyperbolic sine, double complex.

### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX casinh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

## Return value

The inverse hyperbolic sine of x according to the following table:

Argument	$\cosh(\text{Argument})$
+0 + 0i	+0 + 0i
+0 + $\infty i$	$+\infty + \frac{1}{2}\pi i$
a + NaNi	NaN + NaNi
$+\infty + bi$	$+\infty + 0i$ , for positive-signed b
$+\infty + \infty i$	$+\pi + 0i$
$+\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for finite nonzero b
NaN + $\infty i$	$\pm\infty + \text{NaN}i$ , sign of real part unspecified
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\cosh(\text{conj}(z)) = \text{conj}(\cosh(z))$ .

For arguments with a negative real component, use the equality:

- $\cosh(-z) = -\cosh(z)$ .

## *[casinhf\(\)](#)*

## Description

Compute inverse hyperbolic sine, float complex.

## Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX casinhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
-----------	-------------

x Value to compute inverse hyperbolic sineof.

Return value

The inverse hyperbolic sine of x according to the following table:

Argument	casinh(Argument)
+0 + 0i	+0 + 0i
+0 + ∞i	+∞ + $\frac{1}{2}\pi i$
a + NaNi	NaN + NaNi
+∞ + bi	+∞ + 0i, for positive-signed b
+∞ + ∞i	+Pi + 0i
+∞ + NaNi	+∞ + NaNi
NaN + 0i	NaN + 0i
NaN + bi	NaN + NaNi, for finite nonzero b
NaN + ∞i	$\pm\infty + NaNi$ , sign of real part unspecified
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{casinh}(\text{conj}(z)) = \text{conj}(\text{casinh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{casinh}(-z) = -\text{casinh}(z)$ .

*casinhl()*

Description

Compute inverse hyperbolic sine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX casinhl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sineof.

Return value

The inverse hyperbolic sine of x according to the following table:

Argument	$\cosh(\text{Argument})$
$+0 + 0i$	$+0 + 0i$
$+0 + \infty i$	$+\infty + \frac{1}{2}\pi i$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$
$+\infty + bi$	$+\infty + 0i$ , for positive-signed $b$
$+\infty + \infty i$	$+\pi i + 0i$
$+\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + 0i$	$\text{NaN} + 0i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for finite nonzero $b$
$\text{NaN} + \infty i$	$\pm\infty + \text{NaN}i$ , sign of real part unspecified
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\cosh(\text{conj}(z)) = \text{conj}(\cosh(z))$ .

For arguments with a negative real component, use the equality:

- $\cosh(-z) = -\cosh(z)$ .

## *[cacosh\(\)](#)*

### Description

Compute inverse hyperbolic cosine, double complex.

### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cacosh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

### Parameters

Parameter	Description
$x$	Value to compute inverse hyperbolic cosine of.

### Return value

The inverse hyperbolic cosine of  $x$  according to the following table:

Argument	$\cosh(\text{Argument})$
$\pm 0 + 0i$	$+0 + 0i$
$a + \infty i$	$+\infty + \frac{1}{2}\pi i$ , for finite $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite $a$

$-\infty + bi$	$+\infty + \pi i$ , for positive-signed finite b
$+\infty + bi$	$+\infty + 0i$ , for positive-signed finite b
$-\infty + \infty i$	$\pm\infty + \frac{3}{4}\pi i$
$+\infty + \infty i$	$\pm\infty + \frac{1}{4}\pi i$
$\pm\infty + NaNi$	$+\infty + NaNi$
$NaN + bi$	$NaN + NaNi$ , for finite b
$NaN + \infty i$	$+\infty + NaNi$
$NaN + NaNi$	$NaN + NaNi$

For arguments with a negative imaginary component, use the equality:

- $\text{cacosh}(\text{conj}(z)) = \text{conj}(\text{cacosh}(z))$ .

### *cacoshf()*

#### Description

Compute inverse hyperbolic cosine, float complex.

#### Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cacoshf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

#### Return value

The inverse hyperbolic cosine of x according to the following table:

Argument	$\text{cacosh}(\text{Argument})$
$\pm 0 + 0i$	$+0 + 0i$
$a + \infty i$	$+\infty + \frac{1}{2}\pi i$ , for finite a
$a + NaNi$	$NaN + NaNi$ , for finite a
$-\infty + bi$	$+\infty + \pi i$ , for positive-signed finite b
$+\infty + bi$	$+\infty + 0i$ , for positive-signed finite b
$-\infty + \infty i$	$\pm\infty + \frac{3}{4}\pi i$
$+\infty + \infty i$	$\pm\infty + \frac{1}{4}\pi i$
$\pm\infty + NaNi$	$+\infty + NaNi$



$\text{NaN} + bi$        $\text{NaN} + \text{NaN}i$ , for finite  $b$   
 $\text{NaN} + \infty i$        $+\infty + \text{NaN}i$   
 $\text{NaN} + \text{NaN}i$     $\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{cacosh}(\text{conj}(z)) = \text{conj}(\text{cacosh}(z))$ .

*cacoshl()*

Description

Compute inverse hyperbolic cosine, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cacoshl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
$x$	Value to compute inverse hyperbolic cosine of.

Return value

The inverse hyperbolic cosine of  $x$  according to the following table:

Argument	$\text{cacosh}(\text{Argument})$
$\pm 0 + 0i$	$+0 + 0i$
$a + \infty i$	$+\infty + \frac{1}{2}\pi i$ , for finite $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite $a$
$-\infty + bi$	$+\infty + \pi i$ , for positive-signed finite $b$
$+\infty + bi$	$+\infty + 0i$ , for positive-signed finite $b$
$-\infty + \infty i$	$\pm\infty + \frac{3}{4}\pi i$
$+\infty + \infty i$	$\pm\infty + \frac{1}{4}\pi i$
$\pm\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for finite $b$
$\text{NaN} + \infty i$	$+\infty + \text{NaN}i$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{cacosh}(\text{conj}(z)) = \text{conj}(\text{cacosh}(z))$ .

## *catanh()*

### Description

Compute inverse hyperbolic tangent, double complex.

### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX catanh(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

### Return value

The inverse hyperbolic tangent of x according to the following table:

Argument	catanh(Argument)
+0 + 0i	+0 + 0i
+0 + NaNi	+0 + NaNi
+1 + 0i	+∞ + 0i
a + ∞i	+0 + $\frac{1}{2}\pi i$ for positive-signed a
a + NaNi	NaN + NaNi, for nonzero finite a
+∞ + bi	+0 + $\frac{1}{2}\pi i$ for positive-signed b
+∞ + ∞i	+0 + $\frac{1}{2}\pi i$
+∞ + NaNi	+0 + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{catanh}(\text{conj}(z)) = \text{conj}(\text{catanh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{catanh}(-z) = -\text{catanh}(z)$ .

## *catanhf()*

### Description

Compute inverse hyperbolic tangent, float complex.

## Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX catanhf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

## Return value

The inverse hyperbolic tangent of x according to the following table:

Argument	catanh(Argument)
+0 + 0i	+0 + 0i
+0 + NaNi	+0 + NaNi
+1 + 0i	+∞ + 0i
a + ∞i	+0 + $\frac{1}{2}\pi i$ for positive-signed a
a + NaNi	NaN + NaNi, for nonzero finite a
+∞ + bi	+0 + $\frac{1}{2}\pi i$ for positive-signed b
+∞ + ∞i	+0 + $\frac{1}{2}\pi i$
+∞ + NaNi	+0 + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{catanh}(\text{conj}(z)) = \text{conj}(\text{catanh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{catanh}(-z) = -\text{catanh}(z)$ .

## *catanhl()*

## Description

Compute inverse hyperbolic tangent, long double complex.

## Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX catanhl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

The inverse hyperbolic tangent of x according to the following table:

Argument	catanh(Argument)
+0 + 0i	+0 + 0i
+0 + NaNi	+0 + NaNi
+1 + 0i	$+\infty + 0i$
$a + \infty i$	$+0 + \frac{1}{2}\pi i$ for positive-signed a
$a + NaNi$	NaN + NaNi, for nonzero finite a
$+\infty + bi$	$+0 + \frac{1}{2}\pi i$ for positive-signed b
$+\infty + \infty i$	$+0 + \frac{1}{2}\pi i$
$+\infty + NaNi$	+0 + NaNi
NaN + bi	NaN + NaNi, for finite b
NaN + NaNi	NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{catanh}(\text{conj}(z)) = \text{conj}(\text{catanh}(z))$ .

For arguments with a negative real component, use the equality:

- $\text{catanh}(-z) = -\text{catanh}(z)$ .

### Power and absolute value

Function	Description
<a href="#">cabs()</a>	Compute magnitude, double complex.
<a href="#">cabsf()</a>	Compute magnitude, float complex.
<a href="#">cabsl()</a>	Compute magnitude, long double complex.
<a href="#">cpow()</a>	Power, double complex.
<a href="#">cpowf()</a>	Power, float complex.
<a href="#">cpowl()</a>	Power, long double complex.
<a href="#">csqrt()</a>	Square root, double complex.
<a href="#">csqrtf()</a>	Square root, float complex.
<a href="#">csqrtl()</a>	Square root, long double complex.

### *cabs()*

#### Description

Compute magnitude, double complex.

#### Prototype

```
double cabs(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Value to compute magnitude of.

#### Return value

The magnitude of x,  $|x|$ .

### *cabsf()*

#### Description

Compute magnitude, float complex.

#### Prototype

```
float cabsf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

#### Parameters

Parameter	Description
x	Value to compute magnitude of.

#### Return value

The magnitude of x,  $|x|$ .

### *cabs1()*

#### Description

Compute magnitude, long double complex.

#### Prototype

```
long double cabs1(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

#### Parameters

---

Parameter	Description
x	Value to compute magnitude of.

Return value

The magnitude of x, |x|.

*cpow()*

Description

Power, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cpow(__SEGGER_RTL_FLOAT64_C_COMPLEX x,  
                                     __SEGGER_RTL_FLOAT64_C_COMPLEX y);
```

Parameters

---

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power of y.

*cpowf()*

Description

Power, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cpowf(__SEGGER_RTL_FLOAT32_C_COMPLEX x,  
                                       __SEGGER_RTL_FLOAT32_C_COMPLEX y);
```

Parameters

---

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power of y.

*cpowl()*

Description

Power, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cpowl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x,  
                                         __SEGGER_RTL_LDOUBLE_C_COMPLEX y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power of y.

*csqrt()*

Description

Square root, double complex.

Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX csqrt(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute square root of.

Return value

The square root of x according to the following table:

Argument	csqrt(Argument)
$\pm 0 + 0i$	$+0 + 0i$
$a + \infty i$	$+\infty + \infty i$ , for all a
$a + NaNi$	$+NaN + NaNi$ , for finite a

$-\infty + bi$	$+0 + \infty i$ for finite positive-signed $b$
$+\infty + bi$	$+\infty + 0i$ , for finite positive-signed $b$
$+\infty + \infty i$	$+\infty + \frac{1}{4}\pi i$
$-\infty + NaNi$	$+NaN + +/\infty i$ , sign of imaginary part unspecified
$+\infty + NaNi$	$+\infty + NaNi$
$NaN + bi$	$NaN + NaNi$ , for finite $b$
$NaN + \infty i$	$+\infty + \infty i$
$NaN + NaNi$	$NaN + NaNi$

For arguments with a negative imaginary component, use the equality:

- $\text{csqrt}(\text{conj}(z)) = \text{conj}(\text{csqrt}(z))$ .

### *csqrtf()*

#### Description

Square root, float complex.

#### Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX csqrtf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

#### Parameters

Parameter	Description
$x$	Value to compute square root of.

#### Return value

The square root of  $x$  according to the following table:

Argument	$\text{csqrt}(\text{Argument})$
$\pm 0 + 0i$	$+0 + 0i$
$a + \infty i$	$+\infty + \infty i$ , for all $a$
$a + NaNi$	$+NaN + NaNi$ , for finite $a$
$-\infty + bi$	$+0 + \infty i$ for finite positive-signed $b$
$+\infty + bi$	$+\infty + 0i$ , for finite positive-signed $b$
$+\infty + \infty i$	$+\infty + \frac{1}{4}\pi i$
$-\infty + NaNi$	$+NaN + +/\infty i$ , sign of imaginary part unspecified
$+\infty + NaNi$	$+\infty + NaNi$



$\text{NaN} + bi$        $\text{NaN} + \text{NaN}i$ , for finite  $b$   
 $\text{NaN} + \infty i$        $+\infty + \infty i$   
 $\text{NaN} + \text{NaN}i$     $\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{csqrt}(\text{conj}(z)) = \text{conj}(\text{csqrt}(z))$ .

### *csqrtl()*

#### Description

Square root, long double complex.

#### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX csqrtl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

#### Parameters

Parameter	Description
$x$	Value to compute square root of.

#### Return value

The square root of  $x$  according to the following table:

Argument	$\text{csqrt}(\text{Argument})$
$\pm 0 + 0i$	$+0 + 0i$
$a + \infty i$	$+\infty + \infty i$ , for all $a$
$a + \text{NaN}i$	$+\text{NaN} + \text{NaN}i$ , for finite $a$
$-\infty + bi$	$+0 + \infty i$ for finite positive-signed $b$
$+\infty + bi$	$+\infty + 0i$ , for finite positive-signed $b$
$+\infty + \infty i$	$+\infty + \frac{1}{4}\pi i$
$-\infty + \text{NaN}i$	$+\text{NaN} + +/\infty i$ , sign of imaginary part unspecified
$+\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for finite $b$
$\text{NaN} + \infty i$	$+\infty + \infty i$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{csqrt}(\text{conj}(z)) = \text{conj}(\text{csqrt}(z))$ .

## Exponential and logarithm functions

Function	Description
<code>clog()</code>	Compute natural logarithm, double complex.
<code>clogf()</code>	Compute natural logarithm, float complex.
<code>clogl()</code>	Compute natural logarithm, long double complex.
<code>cexp()</code>	Compute base-e exponential, double complex.
<code>cexpf()</code>	Compute base-e exponential, float complex.
<code>cexpl()</code>	Compute base-e exponential, long double complex.

### *clog()*

#### Description

Compute natural logarithm, double complex.

#### Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX clog(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

#### Parameters

Parameter	Description
<code>x</code>	Value to compute logarithm of.

#### Return value

The natural logarithm of `x` according to the following table:

Argument	<code>clog(Argument)</code>
$-0 + 0i$	$-\infty + \pi i$
$+0 + 0i$	$-\infty + 0i$
$a + \infty i$	$+\infty + \frac{1}{2}\pi i$ , for finite $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite $a$
$-\infty + bi$	$+\infty + \pi i$ , for finite positive $b$
$+\infty + bi$	$+\infty + 0i$ , for finite positive $b$
$-\infty + \infty i$	$+\infty + \frac{3}{4}\pi i$
$+\infty + \infty i$	$+\infty + \frac{1}{4}\pi i$
$\pm\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for finite $b$
$\text{NaN} + \infty i$	$+\infty + \text{NaN}i$

NaN + NaNi NaN + NaNi

For arguments with a negative imaginary component, use the equality:

- $\text{clog}(\text{conj}(z)) = \text{conj}(\text{clog}(z))$ .

*clogf()*

Description

Compute natural logarithm, float complex.

Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX clogf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

The natural logarithm of x according to the following table:

Argument	clog(Argument)
-0 + 0i	$-\infty + \pi i$
+0 + 0i	$-\infty + 0i$
$a + \infty i$	$+\infty + \frac{1}{2}\pi i$ , for finite a
$a + \text{NaNi}$	$\text{NaN} + \text{NaNi}$ , for finite a
$-\infty + bi$	$+\infty + \pi i$ , for finite positive b
$+\infty + bi$	$+\infty + 0i$ , for finite positive b
$-\infty + \infty i$	$+\infty + \frac{3}{4}\pi i$
$+\infty + \infty i$	$+\infty + \frac{1}{4}\pi i$
$\pm\infty + \text{NaNi}$	$+\infty + \text{NaNi}$
$\text{NaN} + bi$	$\text{NaN} + \text{NaNi}$ , for finite b
$\text{NaN} + \infty i$	$+\infty + \text{NaNi}$
$\text{NaN} + \text{NaNi}$	$\text{NaN} + \text{NaNi}$

For arguments with a negative imaginary component, use the equality:

- $\text{clog}(\text{conj}(z)) = \text{conj}(\text{clog}(z))$ .

## *clogl()*

### Description

Compute natural logarithm, long double complex.

### Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX clogl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

### Parameters

Parameter	Description
x	Value to compute logarithm of.

### Return value

The natural logarithm of x according to the following table:

Argument	clog(Argument)
-0 + 0i	$-\infty + \pi i$
+0 + 0i	$-\infty + 0i$
$a + \infty i$	$+\infty + \frac{1}{2}\pi i$ , for finite a
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite a
$-\infty + bi$	$+\infty + \pi i$ , for finite positive b
$+\infty + bi$	$+\infty + 0i$ , for finite positive b
$-\infty + \infty i$	$+\infty + \frac{3}{4}\pi i$
$+\infty + \infty i$	$+\infty + \frac{1}{4}\pi i$
$\pm\infty + \text{NaN}i$	$+\infty + \text{NaN}i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for finite b
$\text{NaN} + \infty i$	$+\infty + \text{NaN}i$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality:

- $\text{clog}(\text{conj}(z)) = \text{conj}(\text{clog}(z))$ .

## *cexp()*

### Description

Compute base-e exponential, double complex.

## Prototype

```
__SEGGER_RTL_FLOAT64_C_COMPLEX cexp(__SEGGER_RTL_FLOAT64_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute exponential of.

## Return value

The base-e exponential of  $x=a+bi$  according to the following table:

Argument	cexp(Argument)
$-/-0 + 0i$	$+1 + 0i$
$a + \infty i$	$\text{NaN} + \text{NaN}i$ , for finite a
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite a
$+\infty + 0i$	$+\infty + 0i$ , for finite positive b
$-\infty + bi$	$+0 \text{ cis}(b)$ for finite b
$+\infty + bi$	$+\infty \text{ cis}(b)$ for finite nonzero b
$-\infty + \infty i$	$\pm\infty + \pm 0i$ , signs unspecified
$+\infty + \infty i$	$\pm\infty + i.\text{NaN}$ , sign of real part unspecified
$-\infty + \text{NaN}i$	$\pm 0 + \pm 0i$ , signs unspecified
$+\infty + \text{NaN}i$	$\pm\infty + \text{NaN}i$ , sign of real part unspecified
$\text{NaN} + 0i$	$\text{NaN} + 0i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for nonzero b
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality

- $\text{cexp}(\text{conj}(x)) = \text{conj}(\text{cexp}(x))$ .

## *cexpf()*

## Description

Compute base-e exponential, float complex.

## Prototype

```
__SEGGER_RTL_FLOAT32_C_COMPLEX cexpf(__SEGGER_RTL_FLOAT32_C_COMPLEX x);
```

## Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

The base-e exponential of  $x=a+bi$  according to the following table:

Argument	$\text{cexp}(\text{Argument})$
$-/-0 + 0i$	$+1 + 0i$
$a + \infty i$	NaN + NaNi, for finite a
$a + \text{NaN}i$	NaN + NaNi, for finite a
$+\infty + 0i$	$+\infty + 0i$ , for finite positive b
$-\infty + bi$	$+0 \text{ cis}(b)$ for finite b
$+\infty + bi$	$+\infty \text{ cis}(b)$ for finite nonzero b
$-\infty + \infty i$	$\pm\infty + \pm 0i$ , signs unspecified
$+\infty + \infty i$	$\pm\infty + i.\text{NaN}$ , sign of real part unspecified
$-\infty + \text{NaN}i$	$\pm 0 + \pm 0i$ , signs unspecified
$+\infty + \text{NaN}i$	$\pm\infty + \text{NaN}i$ , sign of real part unspecified
$\text{NaN} + 0i$	NaN + 0i
$\text{NaN} + bi$	NaN + NaNi, for nonzero b
$\text{NaN} + \text{NaN}i$	NaN + NaNi

For arguments with a negative imaginary component, use the equality

- $\text{cexp}(\text{conj}(x)) = \text{conj}(\text{cexp}(x))$ .

*cexpl()*

Description

Compute base-e exponential, long double complex.

Prototype

```
__SEGGER_RTL_LDOUBLE_C_COMPLEX cexpl(__SEGGER_RTL_LDOUBLE_C_COMPLEX x);
```

Parameters

Parameter	Description
x	Value to compute exponential of.

Return value

The base-e exponential of  $x=a+bi$  according to the following table:

Argument	$\text{cexp}(\text{Argument})$
$-/-0 + 0i$	$+1 + 0i$
$a + \infty i$	$\text{NaN} + \text{NaN}i$ , for finite $a$
$a + \text{NaN}i$	$\text{NaN} + \text{NaN}i$ , for finite $a$
$+\infty + 0i$	$+\infty + 0i$ , for finite positive $b$
$-\infty + bi$	$+0 \text{ cis}(b)$ for finite $b$
$+\infty + bi$	$+\infty \text{ cis}(b)$ for finite nonzero $b$
$-\infty + \infty i$	$\pm\infty + \pm 0i$ , signs unspecified
$+\infty + \infty i$	$\pm\infty + i.\text{NaN}$ , sign of real part unspecified
$-\infty + \text{NaN}i$	$\pm 0 + \pm 0i$ , signs unspecified
$+\infty + \text{NaN}i$	$\pm\infty + \text{NaN}i$ , sign of real part unspecified
$\text{NaN} + 0i$	$\text{NaN} + 0i$
$\text{NaN} + bi$	$\text{NaN} + \text{NaN}i$ , for nonzero $b$
$\text{NaN} + \text{NaN}i$	$\text{NaN} + \text{NaN}i$

For arguments with a negative imaginary component, use the equality

- $\text{cexp}(\text{conj}(x)) = \text{conj}(\text{cexp}(x))$ .

## <ctype.h>

### Classification functions

Function	Description
<a href="#">iscntrl()</a>	Is character a control?
<a href="#">iscntrl_l()</a>	Is character a control, per locale? (POSIX.1).
<a href="#">isblank()</a>	Is character a blank?
<a href="#">isblank_l()</a>	Is character a blank, per locale? (POSIX.1).
<a href="#">isspace()</a>	Is character a whitespace character?
<a href="#">isspace_l()</a>	Is character a whitespace character, per locale? (POSIX.1).
<a href="#">ispunct()</a>	Is character a punctuation mark?
<a href="#">ispunct_l()</a>	Is character a punctuation mark, per locale? (POSIX.1).
<a href="#">isdigit()</a>	Is character a decimal digit?
<a href="#">isdigit_l()</a>	Is character a decimal digit, per locale? (POSIX.
<a href="#">isxdigit()</a>	Is character a hexadecimal digit?
<a href="#">isxdigit_l()</a>	Is character a hexadecimal digit, per locale? (POSIX.1).

`isalpha()` Is character alphabetic?  
`isalpha_l()` Is character alphabetic, per locale? (POSIX.1).  
`isalnum()` Is character alphanumeric?  
`isalnum_l()` Is character alphanumeric, per locale? (POSIX.1).  
`isupper()` Is character an uppercase letter?  
`isupper_l()` Is character an uppercase letter, per locale? (POSIX.1).  
`islower()` Is character a lowercase letter?  
`islower_l()` Is character a lowercase letter, per locale? (POSIX.1).  
`isprint()` Is character printable?  
`isprint_l()` Is character printable, per locale? (POSIX.1).  
`isgraph()` Is character any printing character?  
`isgraph_l()` Is character any printing character, per locale? (POSIX.1).

### *iscntrl()*

#### Description

Is character a control?

#### Prototype

```
int iscntrl(int c);
```

#### Parameters

Parameter	Description
c	Character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is a control character in the current locale.

### *iscntrl\_l()*

#### Description

Is character a control, per locale? (POSIX.1).

#### Prototype

```
int iscntrl_l(int c,  
               locale_t loc);
```



## Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

## Return value

Returns nonzero (true) if and only if the value of the argument c is a control character in the locale loc.

## Notes

Conforms to POSIX.1-2017.

[\*isblank\(\)\*](#)

## Description

Is character a blank?

## Prototype

```
int isblank(int c);
```

## Parameters

Parameter	Description
c	Character to test.

## Return value

Returns nonzero (true) if and only if the value of the argument c is either a space character or tab character in the current locale.

[\*isblank\\_l\(\)\*](#)

## Description

Is character a blank, per locale? (POSIX.1).

## Prototype

```
int isblank_l(int c,  
              locale_t loc);
```

## Parameters

---

Parameter	Description
<code>c</code>	Character to test.
<code>loc</code>	Locale used to test <code>c</code> .

#### Return value

Returns nonzero (true) if and only if the value of the argument `c` is either a space character or the tab character in locale `loc`.

#### Notes

Conforms to POSIX.1-2017.

#### *isspace()*

##### Description

Is character a whitespace character?

##### Prototype

```
int isspace(int c);
```

##### Parameters

---

Parameter	Description
<code>c</code>	Character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument `c` is a standard white-space character in the current locale. The standard white-space characters are space, form feed, new-line, carriage return, horizontal tab, and vertical tab.

#### *isspace\_l()*

##### Description

Is character a whitespace character, per locale? (POSIX.1).

##### Prototype

```
int isspace_l(int c,  
              locale_t loc);
```

##### Parameters

---

Parameter	Description
-----------	-------------

---

c	Character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the locale loc.

#### Notes

Conforms to POSIX.1-2017.

#### *ispunct()*

##### Description

Is character a punctuation mark?

##### Prototype

```
int ispunct(int c);
```

##### Parameters

Parameter	Description
c	Character to test.

#### Return value

Returns nonzero (true) for every printing character for which neither `isspace()` nor `isalnum()` is true in the current locale.

#### *ispunct\_l()*

##### Description

Is character a punctuation mark, per locale? (POSIX.1).

##### Prototype

```
int ispunct_l(int c,  
              locale_t loc);
```

##### Parameters

Parameter	Description
c	Character to test.

loc            Locale used to test c.

#### Return value

Returns nonzero (true) for every printing character for which neither `isspace()` nor `isalnum()` is true in the locale loc.

#### Notes

Conforms to POSIX.1-2017.

#### *isdigit()*

##### Description

Is character a decimal digit?

##### Prototype

```
int isdigit(int c);
```

##### Parameters

Parameter	Description
c	Character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the current locale.

#### *isdigit\_l()*

##### Description

Is character a decimal digit, per locale? (POSIX.1)

##### Prototype

```
int isdigit_l(int c,  
              locale_t loc);
```

##### Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument *c* is a digit in the locale *loc*.

Notes

Conforms to POSIX.1-2017.

*isxdigit()*

Description

Is character a hexadecimal digit?

Prototype

```
int isxdigit(int c);
```

Parameters

Parameter	Description
<i>c</i>	Character to test.

Return value

Returns nonzero (true) if and only if the value of the argument *c* is a hexadecimal digit in the current locale.

*isxdigit\_l()*

Description

Is character a hexadecimal digit, per locale? (POSIX.1).

Prototype

```
int isxdigit_l(int c,  
               locale_t loc);
```

Parameters

Parameter	Description
<i>c</i>	Character to test.
<i>loc</i>	Locale used to test <i>c</i> .

Return value

Returns nonzero (true) if and only if the value of the argument *c* is a hexadecimal digit in the current locale.

## Notes

Conforms to POSIX.1-2017.

### *isalpha()*

#### Description

Is character alphabetic?

#### Prototype

```
int isalpha(int c);
```

#### Parameters

Parameter	Description
c	Character to test.

#### Return value

Returns true if the character c is alphabetic in the current locale. That is, any character for which *isupper()* or *islower()* returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of *iscntrl()*, *isdigit()*, *ispunct()*, or *isspace()* is true.

In the C locale, *isalpha()* returns nonzero (true) if and only if *isupper()* or *islower()* return true for value of the argument c.

### *isalpha\_l()*

#### Description

Is character alphabetic, per locale? (POSIX.1).

#### Prototype

```
int isalpha_l(int c,  
              locale_t loc);
```

#### Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

#### Return value

Returns true if the character *c* is alphabetic in the locale *loc*. That is, any character for which `isupper()` or `islower()` returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of `iscntrl()`, `isdigit()`, `ispunct()`, or `isspace()` is true in the locale *loc*.

In the C locale, `isalpha()` returns nonzero (true) if and only if `isupper()` or `islower()` return true for value of the argument *c*.

#### Notes

Conforms to POSIX.1-2017.

#### `isalnum()`

##### Description

Is character alphanumeric?

##### Prototype

```
int isalnum(int c);
```

##### Parameters

Parameter	Description
<i>c</i>	Character to test.

##### Return value

Returns nonzero (true) if and only if the value of the argument *c* is an alphabetic or numeric character in the current locale.

#### `isalnum_l()`

##### Description

Is character alphanumeric, per locale? (POSIX.1).

##### Prototype

```
int isalnum_l(int c,  
              locale_t loc);
```

##### Parameters

Parameter	Description
<i>c</i>	Character to test.
<i>loc</i>	Locale used to test <i>c</i> .

## Return value

Returns nonzero (true) if and only if the value of the argument `c` is an alphabetic or numeric character in the locale `loc`.

## Notes

Conforms to POSIX.1-2017.

## *isupper()*

### Description

Is character an uppercase letter?

### Prototype

```
int isupper(int c);
```

### Parameters

Parameter	Description
<code>c</code>	Character to test.

## Return value

Returns nonzero (true) if and only if the value of the argument `c` is an uppercase letter in the current locale.

## *isupper\_l()*

### Description

Is character an uppercase letter, per locale? (POSIX.1).

### Prototype

```
int isupper_l(int c,  
              locale_t loc);
```

### Parameters

Parameter	Description
<code>c</code>	Character to test.
<code>loc</code>	Locale used to test <code>c</code> .

## Return value



Returns nonzero (true) if and only if the value of the argument `c` is an uppercase letter in the locale `loc`.

#### Notes

Conforms to POSIX.1-2017.

#### *islower()*

##### Description

Is character a lowercase letter?

##### Prototype

```
int islower(int c);
```

##### Parameters

Parameter	Description
<code>c</code>	Character to test.

##### Return value

Returns nonzero (true) if and only if the value of the argument `c` is a lowercase letter in the current locale.

#### *islower\_l()*

##### Description

Is character a lowercase letter, per locale? (POSIX.1).

##### Prototype

```
int islower_l(int c,  
              locale_t loc);
```

##### Parameters

Parameter	Description
<code>c</code>	Character to test.
<code>loc</code>	Locale used to test <code>c</code> .

##### Return value

Returns nonzero (true) if and only if the value of the argument `c` is a lowercase letter in the locale `loc`.

## Notes

Conforms to POSIX.1-2017.

### *isprint()*

#### Description

Is character printable?

#### Prototype

```
int isprint(int c);
```

#### Parameters

Parameter	Description
c	Character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the current locale.

### *isprint\_l()*

#### Description

Is character printable, per locale? (POSIX.1).

#### Prototype

```
int isprint_l(int c,  
              locale_t loc);
```

#### Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the locale loc.

## Notes

Conforms to POSIX.1-2017.

### *isgraph()*

#### Description

Is character any printing character?

#### Prototype

```
int isgraph(int c);
```

#### Parameters

Parameter	Description
c	Character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the current locale.

### *isgraph\_l()*

#### Description

Is character any printing character, per locale? (POSIX.1).

#### Prototype

```
int isgraph_l(int c,  
              locale_t loc);
```

#### Parameters

Parameter	Description
c	Character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the locale loc.

#### Notes

Conforms to POSIX.1-2017.

## Conversion functions

Function	Description
<code>toupper()</code>	Convert lowercase character to uppercase.
<code>toupper_l()</code>	Convert lowercase character to uppercase, per locale (POSIX.1).
<code>tolower()</code>	Convert uppercase character to lowercase.
<code>tolower_l()</code>	Convert uppercase character to lowercase, per locale (POSIX.1).

### `toupper()`

#### Description

Convert lowercase character to uppercase.

#### Prototype

```
int toupper(int c);
```

#### Parameters

Parameter	Description
<code>c</code>	Character to convert.

#### Return value

Converted character.

#### Additional information

Converts a lowercase letter to a corresponding uppercase letter.

If the argument `c` is a character for which `islower()` is true and there are one or more corresponding characters, as specified by the current locale, for which `isupper()` is true, `toupper()` returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

#### Notes

Even though `islower()` can return true for some characters, `toupper()` may return that lowercase character unchanged as there are no corresponding uppercase characters in the locale.

### `toupper_l()`

#### Description

Convert lowercase character to uppercase, per locale (POSIX.1).

#### Prototype

```
int toupper_l(int c,  
              locale_t loc);
```

#### Parameters

Parameter	Description
c	Character to convert.
loc	Locale used to convert c.

#### Return value

Converted character.

#### Additional information

Converts a lowercase letter to a corresponding uppercase letter in locale loc. If the argument c is a character for which `islower_l()` is true in locale loc, `toupper_l()` returns the corresponding uppercase letter; otherwise, the argument is returned unchanged.

#### Notes

Conforms to POSIX.1-2017.

#### `tolower()`

#### Description

Convert uppercase character to lowercase.

#### Prototype

```
int tolower(int c);
```

#### Parameters

Parameter	Description
c	Character to convert.

#### Return value

Converted character.

#### Additional information

Converts an uppercase letter to a corresponding lowercase letter.

If the argument c is a character for which `isupper()` is true and there are one or more corresponding characters, as specified by the current locale, for which `islower()` is true, the `tolower()` function

returns one of the corresponding characters (always the same one for any given locale); otherwise, the argument is returned unchanged.

#### Notes

Even though `isupper()` can return true for some characters, `tolower()` may return that uppercase character unchanged as there are no corresponding lowercase characters in the locale.

#### `tolower_l()`

#### Description

Convert uppercase character to lowercase, per locale (POSIX.1).

#### Prototype

```
int tolower_l(int c,  
              locale_t loc);
```

#### Parameters

Parameter	Description
<code>c</code>	Character to convert.
<code>loc</code>	Locale used to convert <code>c</code> .

#### Return value

Converted character.

#### Additional information

Converts an uppercase letter to a corresponding lowercase letter in locale `loc`. If the argument is a character for which `isupper_l()` is true in locale `loc`, `tolower_l()` returns the corresponding lowercase letter; otherwise, the argument is returned unchanged.

#### Notes

Conforms to POSIX.1-2017.

#### `<errno.h>`

#### Errors

#### Error names

#### Description

Symbolic error names for raised errors.

## Definition

```
#define EDOM      0x01
#define EILSEQ    0x02
#define ERANGE    0x03
#define EHEAP     0x04
#define ENOMEM    0x05
#define EINVAL    0x06
```

## Symbols

Definition	Description
EDOM	Domain error
EILSEQ	Illegal multibyte sequence in conversion
ERANGE	Range error
EHEAP	Heap is corrupt
ENOMEM	Out of memory
EINVAL	Invalid parameter

## *errno*

## Description

Macro returning the current error.

## Definition

```
#define errno    (*__SEGGER_RTL_X_errno_addr())
```

## Additional information

The value in `errno` is significant only when the return value of the call indicated an error. A function that succeeds is allowed to change `errno`. The value of `errno` is never set to zero by a library function.

## <fenv.h>

## Floating-point exceptions

Function	Description
<a href="#">feclearexcept()</a>	Clear floating-point exceptions.
<a href="#">feraiseexcept()</a>	Raise floating-point exceptions.
<a href="#">fegetexceptflag()</a>	Get floating-point exceptions.
<a href="#">fesetexceptflag()</a>	Set floating-point exceptions.

---

`fetestexcept()` Test floating-point exceptions.

### `feclearexcept()`

#### Description

Clear floating-point exceptions.

#### Prototype

```
int feclearexcept(int excepts);
```

#### Parameters

Parameter	Description
<code>excepts</code>	Bitmask of floating-point exceptions to clear.

#### Return value

- `= 0` Floating-point exceptions successfully cleared.
- `≠ 0` Floating-point exceptions not cleared or not supported.

#### Additional information

This function attempts to clear the floating-point exceptions indicated by `excepts`.

#### Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

### `feraiseexcept()`

#### Description

Raise floating-point exceptions.

#### Prototype

```
int feraiseexcept(int excepts);
```

#### Parameters

Parameter	Description
<code>excepts</code>	Bitmask of floating-point exceptions to raise.

#### Return value

- `= 0` All floating-point exceptions successfully raised.
-



≠ 0 Floating-point exceptions not successfully raised or not supported.

#### Additional information

This function attempts to raise the floating-point exceptions indicated by `excepts`.

#### Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

#### [fegetexceptflag\(\)](#)

##### Description

Get floating-point exceptions.

##### Prototype

```
int fegetexceptflag(fexcept_t * flagp,  
                   int         excepts);
```

##### Parameters

Parameter	Description
<code>flagp</code>	Pointer to object that receives the floating-point exception state.
<code>excepts</code>	Bitmask of floating-point exceptions to store.

##### Return value

= 0 Floating-point exceptions correctly stored.

≠ 0 Floating-point exceptions not correctly stored.

#### Additional information

This function attempts to save the floating-point exceptions indicated by `excepts` to the object pointed to by `flagp`.

#### See also

[fesetexceptflag\(\)](#).

#### [fesetexceptflag\(\)](#)

##### Description

Set floating-point exceptions.

##### Prototype

```
int fesetexceptflag(const fexcept_t * flagp,  
                    int             excepts);
```

#### Parameters

Parameter	Description
flagp	Pointer to object containing a previously-stored floating-point exception state.
excepts	Bitmask of floating-point exceptions to restore.

#### Return value

- = 0 Floating-point exceptions correctly restored.
- ≠ 0 Floating-point exceptions not correctly restored.

#### Additional information

This function attempts to restore the floating-point exceptions indicated by `excepts` from the object pointed to by `flagp`. The exceptions to restore as indicated by `excepts` must have at least been specified when storing the exceptions using [fegetexceptflag\(\)](#).

#### See also

[fegetexceptflag\(\)](#).

[fetestexcept\(\)](#)

#### Description

Test floating-point exceptions.

#### Prototype

```
int fetestexcept(int excepts);
```

#### Parameters

Parameter	Description
excepts	Bitmask of floating-point exceptions to test.

#### Return value

The bitmask of all floating-point exceptions that are currently set and are specified in `excepts`.

#### Additional information

This function determines which of the floating-point exceptions indicated by `excepts` are currently set.

## Floating-point rounding mode

Function	Description
<a href="#">fegetround()</a>	Get floating-point rounding mode.
<a href="#">fesetround()</a>	Set floating-point rounding mode.

### [fegetround\(\)](#)

#### Description

Get floating-point rounding mode.

#### Prototype

```
int fegetround(void);
```

#### Return value

- $\geq 0$  Current floating-point rounding mode.
- $< 0$  Floating-point rounding mode cannot be determined.

#### Additional information

This function attempts to read the current floating-point rounding mode.

See also

[fesetround\(\)](#).

### [fesetround\(\)](#)

#### Description

Set floating-point rounding mode.

#### Prototype

```
int fesetround(int round);
```

#### Parameters

Parameter	Description
round	Rounding mode to set.

#### Return value

- $= 0$  Current floating-point rounding mode is set to round.
- $\neq 0$  Requested floating-point rounding mode cannot be set.

## Additional information

This function attempts to set the current floating-point rounding mode to round.

See also

[fegetround\(\)](#).

## Floating-point environment

Function	Description
<a href="#">fegetenv()</a>	Get floating-point environment.
<a href="#">fesetenv()</a>	Set floating-point environment.
<a href="#">feupdateenv()</a>	Restore floating-point environment and reraise exceptions.
<a href="#">feholdexcept()</a>	Save floating-point environment and set non-stop mode.

### *fegetenv()*

#### Description

Get floating-point environment.

#### Prototype

```
int fegetenv(fenv_t * envp);
```

#### Parameters

Parameter	Description
<code>envp</code>	Pointer to object that receives the floating-point environment.

#### Return value

- `= 0` Current floating-point environment successfully stored.
- `≠ 0` Floating-point environment cannot be stored.

## Additional information

This function attempts to store the current floating-point environment to the object pointed to by `envp`.

#### Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

See also

[fesetenv\(\)](#).

[fesetenv\(\)](#)

## Description

Set floating-point environment.

## Prototype

```
int fesetenv(const fenv_t * envp);
```

## Parameters

Parameter	Description
envp	Pointer to object containing previously-stored floating-point environment.

## Return value

- = 0 Current floating-point environment successfully restored.
- ≠ 0 Floating-point environment cannot be restored.

## Additional information

This function attempts to restore the floating-point environment from the object pointed to by envp.

## Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

## See also

[fegetenv\(\)](#).

[feupdateenv\(\)](#)

## Description

Restore floating-point environment and reraise exceptions.

## Prototype

```
int feupdateenv(const fenv_t * envp);
```

## Parameters

Parameter	Description
envp	Pointer to object containing previously-stored floating-point environment.

## Return value

- = 0 Environment restored and exceptions raised successfully.
- ≠ 0 Failed to restore environment and raise exceptions.

## Additional information

This function attempts to save the currently raised floating-point exceptions, restore the floating-point environment from the object pointed to by `envp`, and raise the saved exceptions.

## Notes

This function has no return value in ISO C (1999) and an integer return value in ISO C (2008).

## *[feholdexcept\(\)](#)*

## Description

Save floating-point environment and set non-stop mode.

## Prototype

```
int feholdexcept(fenv_t * envp);
```

## Parameters

Parameter	Description
<code>envp</code>	Pointer to object that receives the floating-point environment.

## Return value

- = 0 Environment stored and non-stop mode set successfully.
- ≠ 0 Failed to store environment or set non-stop mode.

## Additional information

This function saves the current floating-point environment to the object pointed to by `envp`, clears the floating-point status flags, and then installs a non-stop mode for all floating-point exceptions.

## **<float.h>**

## Floating-point constants

## *Common parameters*

## Description

Applies to single-precision and double-precision formats.

#### Definition

```
#define FLT_ROUNDS          1
#define FLT_EVAL_METHOD    0
#define FLT_RADIX          2
#define DECIMAL_DIG        17
```

#### Symbols

Definition	Description
FLT_ROUNDS	Rounding mode of floating-point addition is round to nearest.
FLT_EVAL_METHOD	All operations and constants are evaluated to the range and precision of the type.
FLT_RADIX	Radix of the exponent representation.
DECIMAL_DIG	Number of decimal digits that can be rounded to a floating-point number without change to the value.

### Float parameters

#### Description

IEEE 32-bit single-precision floating format parameters.

#### Definition

```
#define FLT_MANT_DIG        24
#define FLT_EPSILON        1.19209290E-07f
#define FLT_DIG            6
#define FLT_MIN_EXP        -125
#define FLT_MIN            1.17549435E-38f
#define FLT_MIN_10_EXP     -37
#define FLT_MAX_EXP        +128
#define FLT_MAX            3.40282347E+38f
#define FLT_MAX_10_EXP     +38
```

#### Symbols

Definition	Description
FLT_MANT_DIG	Number of base FLT_RADIX digits in the mantissa part of a float.
FLT_EPSILON	Minimum positive number such that $1.0f + \text{FLT\_EPSILON} \neq 1.0f$ .
FLT_DIG	Number of decimal digits of precision of a float.
FLT_MIN_EXP	Minimum value of base FLT_RADIX in the exponent part of a float.

FLT_MIN	Minimum value of a float.
FLT_MIN_10_EXP	Minimum value in base 10 of the exponent part of a float.
FLT_MAX_EXP	Maximum value of base FLT_RADIX in the exponent part of a float.
FLT_MAX	Maximum value of a float.
FLT_MAX_10_EXP	Maximum value in base 10 of the exponent part of a float.

### Double parameters

#### Description

IEEE 64-bit double-precision floating format parameters.

#### Definition

```
#define DBL_MANT_DIG      53
#define DBL_EPSILON      2.2204460492503131E-16
#define DBL_DIG          15
#define DBL_MIN_EXP      -1021
#define DBL_MIN          2.2250738585072014E-308
#define DBL_MIN_10_EXP   -307
#define DBL_MAX_EXP      +1024
#define DBL_MAX          1.7976931348623157E+308
#define DBL_MAX_10_EXP   +308
```

#### Symbols

Definition	Description
DBL_MANT_DIG	Number of base DBL_RADIX digits in the mantissa part of a double.
DBL_EPSILON	Minimum positive number such that $1.0 + \text{DBL\_EPSILON} \neq 1.0$ .
DBL_DIG	Number of decimal digits of precision of a double.
DBL_MIN_EXP	Minimum value of base DBL_RADIX in the exponent part of a double.
DBL_MIN	Minimum value of a double.
DBL_MIN_10_EXP	Minimum value in base 10 of the exponent part of a double.
DBL_MAX_EXP	Maximum value of base DBL_RADIX in the exponent part of a double.
DBL_MAX	Maximum value of a double.
DBL_MAX_10_EXP	Maximum value in base 10 of the exponent part of a double.

### <iso646.h>

The header <iso646.h> defines macros that expand to the corresponding tokens to ease writing C programs with keyboards that do not have keys for frequently-used operators.



## Macros

### *Replacement macros*

#### Description

Standard replacement macros.

#### Definition

```
#define and      &&
#define and_eq   &=
#define bitand   &
#define bitor    |
#define compl    ~
#define not      !
#define not_eq   !=
#define or       ||
#define or_eq    |=
#define xor      ^
#define xor_eq   ^=
```

## <limits.h>

### Minima and maxima

#### *Character minima and maxima*

#### Description

Minimum and maximum values for character types.

#### Definition

```
#define CHAR_BIT      8
#define CHAR_MIN      0
#define CHAR_MAX      255
#define SCHAR_MAX     127
#define SCHAR_MIN     (-128)
#define UCHAR_MAX     255
```

#### Symbols

Definition	Description
CHAR_BIT	Number of bits for smallest object that is not a bit-field (byte).
CHAR_MIN	Minimum value of a plain character.
CHAR_MAX	Maximum value of a plain character.

SCHAR\_MAX Maximum value of a signed character.  
SCHAR\_MIN Minimum value of a signed character.  
UCHAR\_MAX Maximum value of an unsigned character.

### *Short integer minima and maxima*

#### Description

Minimum and maximum values for short integer types.

#### Definition

```
#define SHRT_MIN    (-32767 - 1)
#define SHRT_MAX    32767
#define USHRT_MAX   65535
```

#### Symbols

Definition	Description
SHRT_MIN	Minimum value of a short integer.
SHRT_MAX	Maximum value of a short integer.
USHRT_MAX	Maximum value of an unsigned short integer.

### *Integer minima and maxima*

#### Description

Minimum and maximum values for integer types.

#### Definition

```
#define INT_MIN      (-2147483647 - 1)
#define INT_MAX      2147483647
#define UINT_MAX     4294967295u
```

#### Symbols

Definition	Description
INT_MIN	Minimum value of an integer.
INT_MAX	Maximum value of an integer.
UINT_MAX	Maximum value of an unsigned integer.

### *Long integer minima and maxima (32-bit)*

#### Description

Minimum and maximum values for long integer types.

#### Definition

```
#define LONG_MIN      (-2147483647L - 1)
#define LONG_MAX      2147483647L
#define ULONG_MAX     4294967295uL
```

#### Symbols

Definition	Description
LONG_MIN	Maximum value of a long integer.
LONG_MAX	Minimum value of a long integer.
ULONG_MAX	Maximum value of an unsigned long integer.

#### *Long integer minima and maxima (64-bit)*

#### Description

Minimum and maximum values for long integer types.

#### Definition

```
#define LONG_MIN      (-9223372036854775807L - 1)
#define LONG_MAX      9223372036854775807L
#define ULONG_MAX     18446744073709551615uL
```

#### Symbols

Definition	Description
LONG_MIN	Minimum value of a long integer.
LONG_MAX	Maximum value of a long integer.
ULONG_MAX	Maximum value of an unsigned long integer.

#### *Long long integer minima and maxima*

#### Description

Minimum and maximum values for long integer types.

#### Definition

```
#define LLONG_MIN      (-9223372036854775807LL - 1)
#define LLONG_MAX      9223372036854775807LL
#define ULLONG_MAX     18446744073709551615uLL
```

## Symbols

Definition	Description
LLONG_MIN	Minimum value of a long long integer.
LLONG_MAX	Maximum value of a long long integer.
ULLONG_MAX	Maximum value of an unsigned long long integer.

## Multibyte characters

### Description

Maximum number of bytes in a multi-byte character.

### Definition

```
#define MB_LEN_MAX    4
```

## Symbols

Definition	Description
MB_LEN_MAX	Maximum

### Additional information

The maximum number of bytes in a multi-byte character for any supported locale. Unicode (ISO 10646) characters between 0x000000 and 0x10FFFF inclusive are supported which convert to a maximum of four bytes in the UTF-8 encoding.

## <locale.h>

### Data types

#### `__SEGGER_RTL_lconv`

### Type definition

```
typedef struct {  
    char * decimal_point;  
    char * thousands_sep;  
    char * grouping;  
    char * int_curr_symbol;  
    char * currency_symbol;  
    char * mon_decimal_point;  
    char * mon_thousands_sep;  
    char * mon_grouping;  
    char * positive_sign;
```

```

char * negative_sign;
char  int_frac_digits;
char  frac_digits;
char  p_cs_precedes;
char  p_sep_by_space;
char  n_cs_precedes;
char  n_sep_by_space;
char  p_sign_posn;
char  n_sign_posn;
char  int_p_cs_precedes;
char  int_n_cs_precedes;
char  int_p_sep_by_space;
char  int_n_sep_by_space;
char  int_p_sign_posn;
char  int_n_sign_posn;
} __SEGGER_RTL_lconv;

```

#### Structure members

Member	Description
decimal_point	Decimal point separator.
thousands_sep	Separators used to delimit groups of digits to the left of the decimal point for non-monetary quantities.
grouping	Specifies the amount of digits that form each of the groups to be separated by thousands_sep separator for non-monetary quantities.
int_curr_symbol	International currency symbol.
currency_symbol	Local currency symbol.
mon_decimal_point	Decimal-point separator used for monetary quantities.
mon_thousands_sep	Separators used to delimit groups of digits to the left of the decimal point for monetary quantities.
mon_grouping	Specifies the amount of digits that form each of the groups to be separated by mon_thousands_sep separator for monetary quantities.
positive_sign	Sign to be used for nonnegative (positive or zero) monetary quantities.
negative_sign	Sign to be used for negative monetary quantities.
int_frac_digits	Amount of fractional digits to the right of the decimal point for monetary quantities in the international format.
frac_digits	Amount of fractional digits to the right of the decimal point for monetary quantities in the local format.
p_cs_precedes	Whether the currency symbol should precede nonnegative (positive or zero) monetary quantities.

<code>p_sep_by_space</code>	Whether a space should appear between the currency symbol and nonnegative (positive or zero) monetary quantities.
<code>n_cs_precedes</code>	Whether the currency symbol should precede negative monetary quantities.
<code>n_sep_by_space</code>	Whether a space should appear between the currency symbol and negative monetary quantities.
<code>p_sign_posn</code>	Position of the sign for nonnegative (positive or zero) monetary quantities.
<code>n_sign_posn</code>	Position of the sign for negative monetary quantities.
<code>int_p_cs_precedes</code>	Whether <code>int_curr_symbol</code> precedes or succeeds the value for a nonnegative internationally formatted monetary quantity.
<code>int_n_cs_precedes</code>	Whether <code>int_curr_symbol</code> precedes or succeeds the value for a negative internationally formatted monetary quantity.
<code>int_p_sep_by_space</code>	Value indicating the separation of the <code>int_curr_symbol</code> , the sign string, and the value for a nonnegative internationally formatted monetary quantity.
<code>int_n_sep_by_space</code>	Value indicating the separation of the <code>int_curr_symbol</code> , the sign string, and the value for a negative internationally formatted monetary quantity.
<code>int_p_sign_posn</code>	Value indicating the positioning of the <code>positive_sign</code> for a nonnegative internationally formatted monetary quantity.
<code>int_n_sign_posn</code>	Value indicating the positioning of the <code>positive_sign</code> for a negative internationally formatted monetary quantity.

## Locale management

Function	Description
<code>setlocale()</code>	Set locale.
<code>localeconv()</code>	Get current locale data.

### `setlocale()`

#### Description

Set locale.

#### Prototype

```
char *setlocale(    int    category,
                  const char * loc);
```

#### Parameters

Parameter	Description
<code>category</code>	Category of locale to set, see below.

`loc`            Pointer to name of locale to set or, if NULL, the current locale.

Return value

Returns the name of the current locale.

Additional information

The category parameter can have the following values:

Value	Description
LC_ALL	Entire locale.
LC_COLLATE	Affects <code>strcoll()</code> and <code>strxfrm()</code> .
LC_CTYPE	Affects character handling.
LC_MONETARY	Affects monetary formatting information.
LC_NUMERIC	Affects decimal-point character in I/O and string formatting operations.
LC_TIME	Affects <code>strftime()</code> .

### `localeconv()`

Description

Get current locale data.

Prototype

```
localeconv(void);
```

Return value

Returns a pointer to a structure of type `lconv` with the corresponding values for the current locale filled in.

## `<math.h>`

### Exponential and logarithm functions

Function	Description
<code>sqrt()</code>	Compute square root, double.
<code>sqrtf()</code>	Compute square root, float.
<code>sqrtl()</code>	Compute square root, long double.
<code>cbrt()</code>	Compute cube root, double.
<code>cbrtf()</code>	Compute cube root, float.
<code>cbrtl()</code>	Compute cube root, long double.

---

<code>rsqrt()</code>	Compute reciprocal square root, double.
<code>rsqrtf()</code>	Compute reciprocal square root, float.
<code>rsqrtl()</code>	Compute reciprocal square root, long double.
<code>exp()</code>	Compute base-e exponential, double.
<code>expf()</code>	Compute base-e exponential, float.
<code>expl()</code>	Compute base-e exponential, long double.
<code>expm1()</code>	Compute base-e exponential, modified, double.
<code>expm1f()</code>	Compute base-e exponential, modified, float.
<code>expm1l()</code>	Compute base-e exponential, modified, long double.
<code>exp2()</code>	Compute base-2 exponential, double.
<code>exp2f()</code>	Compute base-2 exponential, float.
<code>exp2l()</code>	Compute base-2 exponential, long double.
<code>exp10()</code>	Compute base-10 exponential, double.
<code>exp10f()</code>	Compute base-10 exponential, float.
<code>exp10l()</code>	Compute base-10 exponential, long double.
<code>frexp()</code>	Split to significand and exponent, double.
<code>frexpf()</code>	Split to significand and exponent, float.
<code>frexpl()</code>	Split to significand and exponent, long double.
<code>hypot()</code>	Compute magnitude of complex, double.
<code>hypotf()</code>	Compute magnitude of complex, float.
<code>hypotl()</code>	Compute magnitude of complex, long double.
<code>log()</code>	Compute natural logarithm, double.
<code>logf()</code>	Compute natural logarithm, float.
<code>logl()</code>	Compute natural logarithm, long double.
<code>log2()</code>	Compute base-2 logarithm, double.
<code>log2f()</code>	Compute base-2 logarithm, float.
<code>log2l()</code>	Compute base-2 logarithm, long double.
<code>log10()</code>	Compute common logarithm, double.
<code>log10f()</code>	Compute common logarithm, float.
<code>log10l()</code>	Compute common logarithm, long double.
<code>logb()</code>	Radix-independent exponent, double.
<code>logbf()</code>	Radix-independent exponent, float.
<code>logbl()</code>	Radix-independent exponent, long double.

---



<code>ilogb()</code>	Radix-independent exponent, double.
<code>ilogbf()</code>	Radix-independent exponent, float.
<code>ilogbl()</code>	Radix-independent exponent, long double.
<code>log1p()</code>	Compute natural logarithm plus one, double.
<code>log1pf()</code>	Compute natural logarithm plus one, float.
<code>log1pl()</code>	Compute natural logarithm plus one, long double.
<code>ldexp()</code>	Scale by power of two, double.
<code>ldexpf()</code>	Scale by power of two, float.
<code>ldexpl()</code>	Scale by power of two, long double.
<code>pow()</code>	Raise to power, double.
<code>powf()</code>	Raise to power, float.
<code>powl()</code>	Raise to power, long double.
<code>scalbn()</code>	Scale, double.
<code>scalbnf()</code>	Scale, float.
<code>scalbnl()</code>	Scale, long double.
<code>scalbln()</code>	Scale, double.
<code>scalblnf()</code>	Scale, float.
<code>scalblnl()</code>	Scale, long double.

## *sqrt()*

### Description

Compute square root, double.

### Prototype

```
double sqrt(double x);
```

### Parameters

Parameter	Description
x	Value to compute square root of.

### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If  $x < 0$ , return NaN.

- Else, return square root of x.

#### Additional information

`sqrt()` computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, `sqrt()` deviates and always uses IEC 60559 semantics.

#### `sqrtf()`

##### Description

Compute square root, float.

##### Prototype

```
float sqrtf(float x);
```

##### Parameters

Parameter	Description
x	Value to compute square root of.

##### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If  $x < 0$ , return NaN.
- Else, return square root of x.

#### Additional information

`sqrt()` computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, `sqrt()` deviates and always uses IEC 60559 semantics.

#### `sqrtl()`

##### Description

Compute square root, long double.

##### Prototype

```
long double sqrtl(long double x);
```

##### Parameters

Parameter	Description
-----------	-------------

x                    Value to compute square root of.

#### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- If  $x < 0$ , return NaN.
- Else, return square root of x.

#### Additional information

`sqrtl()` computes the nonnegative square root of x. C90 and C99 require that a domain error occurs if the argument is less than zero, `sqrtl()` deviates and always uses IEC 60559 semantics.

#### `cbtrl()`

##### Description

Compute cube root, double.

##### Prototype

```
double cbtrl(double x);
```

##### Parameters

Parameter	Description
x	Value to compute cube root of.

#### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

#### `cbtrf()`

##### Description

Compute cube root, float.

##### Prototype

```
float cbtrf(float x);
```

## Parameters

Parameter	Description
x	Value to compute cube root of.

## Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

## *cbtrl()*

## Description

Compute cube root, long double.

## Prototype

```
long double cbtrl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute cube root of.

## Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return cube root of x.

## *rsqrt()*

## Description

Compute reciprocal square root, double.

## Prototype

```
double rsqrt(double x);
```

## Parameters

Parameter	Description
-----------	-------------

x                    Value to compute reciprocal square root of.

Return value

- If x is +/-zero, return +/-infinity.
- If x is positively infinite, return 0.
- If x is NaN, return x.
- If  $x < 0$ , return NaN.
- Else, return reciprocal square root of x.

*rsqrtf()*

Description

Compute reciprocal square root, float.

Prototype

```
float rsqrtf(float x);
```

Parameters

Parameter	Description
x	Value to compute reciprocal square root of.

Return value

- If x is +/-zero, return +/-infinity.
- If x is positively infinite, return 0.
- If x is NaN, return x.
- If  $x < 0$ , return NaN.
- Else, return reciprocal square root of x.

*rsqrtl()*

Description

Compute reciprocal square root, long double.

Prototype

```
long double rsqrtl(long double x);
```

Parameters

Parameter	Description
-----------	-------------

x                    Value to compute reciprocal square root of.

Return value

- If x is +/-zero, return +/-infinity.
- If x is positively infinite, return 0.
- If x is NaN, return x.
- If  $x < 0$ , return NaN.
- Else, return reciprocal square root of x.

*exp()*

Description

Compute base-e exponential, double.

Prototype

```
double exp(double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

*expf()*

Description

Compute base-e exponential, float.

Prototype

```
float expf(float x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

#### Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

#### *expl()*

#### Description

Compute base-e exponential, long double.

#### Prototype

```
long double expl(long double x);
```

#### Parameters

Parameter	Description
x	Value to compute base-e exponential of.

#### Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

#### *expm1()*

#### Description

Compute base-e exponential, modified, double.

#### Prototype

```
double expm1(double x);
```

#### Parameters

Parameter	Description
x	Value to compute exponential of.

#### Return value

- If x is NaN, return x.

- Else, return base-e exponential of x minus 1 ( $e^{**}x - 1$ ).

### *expm1f()*

#### Description

Compute base-e exponential, modified, float.

#### Prototype

```
float expm1f(float x);
```

#### Parameters

Parameter	Description
x	Value to compute exponential of.

#### Return value

- If x is NaN, return x.
- Else, return base-e exponential of x minus 1 ( $e^{**}x - 1$ ).

### *expm1l()*

#### Description

Compute base-e exponential, modified, long double.

#### Prototype

```
long double expm1l(long double x);
```

#### Parameters

Parameter	Description
x	Value to compute exponential of.

#### Return value

- If x is NaN, return x.
- Else, return base-e exponential of x minus 1 ( $e^{**}x - 1$ ).

### *exp2()*

#### Description

Compute base-2 exponential, double.



## Prototype

```
double exp2(double x);
```

## Parameters

Parameter	Description
x	Value to compute base-2 exponential of.

## Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

## *exp2f()*

## Description

Compute base-2 exponential, float.

## Prototype

```
float exp2f(float x);
```

## Parameters

Parameter	Description
x	Value to compute base-e exponential of.

## Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

## *exp2l()*

## Description

Compute base-2 exponential, long double.

## Prototype

```
long double exp2l(long double x);
```

## Parameters

Parameter	Description
x	Value to compute base-2 exponential of.

## Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

## *exp10()*

## Description

Compute base-10 exponential, double.

## Prototype

```
double exp10(double x);
```

## Parameters

Parameter	Description
x	Value to compute base-e exponential of.

## Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

## *exp10f()*

## Description

Compute base-10 exponential, float.

## Prototype

```
float exp10f(float x);
```

## Parameters

Parameter	Description
-----------	-------------

x                    Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

*exp10l()*

Description

Compute base-10 exponential, long double.

Prototype

```
long double exp10l(long double x);
```

Parameters

Parameter	Description
x	Value to compute base-e exponential of.

Return value

- If x is NaN, return x.
- If x is positively infinite, return x.
- If x is negatively infinite, return 0.
- Else, return base-e exponential of x.

*frexp()*

Description

Split to significand and exponent, double.

Prototype

```
double frexp(double x,  
              int * exp);
```

Parameters

Parameter	Description
x	Floating value to operate on.

exp            Pointer to integer receiving the power-of-two exponent of x.

#### Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals  $f * \text{pow}(2, *exp)$

#### Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

#### *frexpf()*

#### Description

Split to significand and exponent, float.

#### Prototype

```
float frexpf(float  x,
              int   * exp);
```

#### Parameters

Parameter	Description
x	Floating value to operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

#### Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals  $f * \text{pow}(2, *exp)$

#### Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

#### *frexpl()*

#### Description

Split to significand and exponent, long double.

#### Prototype

```
long double frexpl(long double x,  
                  int * exp);
```

#### Parameters

Parameter	Description
x	Floating value to operate on.
exp	Pointer to integer receiving the power-of-two exponent of x.

#### Return value

- If x is zero, infinite or NaN, return x and store zero into the integer pointed to by exp.
- Else, return the value f, such that f has a magnitude in the interval [0.5, 1) and x equals f \* pow(2, \*exp)

#### Additional information

Breaks a floating-point number into a normalized fraction and an integral power of two.

#### *hypot()*

#### Description

Compute magnitude of complex, double.

#### Prototype

```
double hypot(double x,  
             double y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return sqrt(x\*x + y\*y).

#### Additional information

Computes the square root of the sum of the squares of  $x$  and  $y$  without undue overflow or underflow. If  $x$  and  $y$  are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

### *hypotf()*

#### Description

Compute magnitude of complex, float.

#### Prototype

```
float hypotf(float x,  
            float y);
```

#### Parameters

Parameter	Description
$x$	Value #1.
$y$	Value #2.

#### Return value

- If  $x$  or  $y$  are infinite, return infinity.
- If  $x$  or  $y$  is NaN, return NaN.
- Else, return  $\sqrt{x^2 + y^2}$ .

#### Additional information

Computes the square root of the sum of the squares of  $x$  and  $y$  without undue overflow or underflow. If  $x$  and  $y$  are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

### *hypotl()*

#### Description

Compute magnitude of complex, long double.

#### Prototype

```
long double hypotl(long double x,  
                  long double y);
```

#### Parameters

Parameter	Description
-----------	-------------

x            Value #1.  
y            Value #2.

Return value

- If x or y are infinite, return infinity.
- If x or y is NaN, return NaN.
- Else, return  $\text{sqrtl}(x*x + y*y)$ .

Additional information

Computes the square root of the sum of the squares of x and y without undue overflow or underflow. If x and y are the lengths of the sides of a right-angled triangle, then this computes the length of the hypotenuse.

*log()*

Description

Compute natural logarithm, double.

Prototype

```
double log(double x);
```

Parameters

Parameter	Description
x	Value to compute logarithm of.

Return value

- If x = NaN, return x.
- If x < 0, return NaN.
- If x = 0, return  $-\infty$ .
- If x is  $+\infty$ , return  $+\infty$ .
- Else, return base-e logarithm of x.

*logf()*

Description

Compute natural logarithm, float.

Prototype

```
float logf(float x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-e logarithm of  $x$ .

## *logl()*

## Description

Compute natural logarithm, long double.

## Prototype

```
long double logl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return  $-\infty$ .
- If  $x$  is  $+\infty$ , return  $+\infty$ .
- Else, return base-e logarithm of  $x$ .

## *log2()*

## Description

Compute base-2 logarithm, double.

## Prototype

```
double log2(double x);
```



## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-10 logarithm of  $x$ .

## *log2f()*

## Description

Compute base-2 logarithm, float.

## Prototype

```
float log2f(float x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-10 logarithm of  $x$ .

## *log2l()*

## Description

Compute base-2 logarithm, long double.

## Prototype

```
long double log2l(long double x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-10 logarithm of  $x$ .

## *log10()*

### Description

Compute common logarithm, double.

### Prototype

```
double log10(double x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-10 logarithm of  $x$ .

## *log10f()*

### Description

Compute common logarithm, float.

### Prototype

```
float log10f(float x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-10 logarithm of  $x$ .

## *log10l()*

## Description

Compute common logarithm, long double.

## Prototype

```
long double log10l(long double x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-10 logarithm of  $x$ .

## *logb()*

## Description

Radix-independent exponent, double.

## Prototype

```
double logb(double x);
```

## Parameters

Parameter	Description
x	Floating value to operate on.

## Return value

- If x is zero, return  $-\infty$ .
- If x is infinite, return  $+\infty$ .
- If x is NaN, return NaN.
- Else, return integer part of  $\log_{\text{FLTRADIX}}(x)$ .

## Additional information

Calculates the exponent of x, which is the integral part of the FLTRADIX-logarithm of x.

*logbf()*

## Description

Radix-independent exponent, float.

## Prototype

```
float logbf(float x);
```

## Parameters

Parameter	Description
x	Floating value to operate on.

## Return value

- If x is zero, return  $-\infty$ .
- If x is infinite, return  $+\infty$ .
- If x is NaN, return NaN.
- Else, return integer part of  $\log_{\text{FLTRADIX}}(x)$ .

## Additional information

Calculates the exponent of x, which is the integral part of the FLTRADIX-logarithm of x.

*logbl()*

## Description

Radix-independent exponent, long double.

## Prototype

```
long double logbl(long double x);
```

## Parameters

Parameter	Description
x	Floating value to operate on.

## Return value

- If x is zero, return  $-\infty$ .
- If x is infinite, return  $+\infty$ .
- If x is NaN, return NaN.
- Else, return integer part of  $\log_{\text{FLTRADIX}}(x)$ .

## Additional information

Calculates the exponent of x, which is the integral part of the FLTRADIX-logarithm of x.

## *ilogb()*

## Description

Radix-independent exponent, double.

## Prototype

```
int ilogb(double x);
```

## Parameters

Parameter	Description
x	Floating value to operate on.

## Return value

- If x is zero, return FP\_ILOGB0.
- If x is NaN, return FP\_ILOGBNAN.
- If x is infinite, return MAX\_INT.
- Else, return integer part of  $\log_{\text{FLTRADIX}}(x)$ .

## *ilogbf()*

## Description

Radix-independent exponent, float.

## Prototype

```
int ilogbf(float x);
```

## Parameters

Parameter	Description
x	Floating value to operate on.

## Return value

- If x is zero, return FP\_ILOGB0.
- If x is NaN, return FP\_ILOGBNAN.
- If x is infinite, return MAX\_INT.
- Else, return integer part of  $\log_{\text{FLT\_RADIX}}(x)$ .

## *ilogbl()*

## Description

Radix-independent exponent, long double.

## Prototype

```
int ilogbl(long double x);
```

## Parameters

Parameter	Description
x	Floating value to operate on.

## Return value

- If x is zero, return FP\_ILOGB0.
- If x is NaN, return FP\_ILOGBNAN.
- If x is infinite, return MAX\_INT.
- Else, return integer part of  $\log_{\text{FLT\_RADIX}}(x)$ .

## *log1p()*

## Description

Compute natural logarithm plus one, double.

## Prototype

```
double log1p(double x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-e logarithm of  $x+1$ .

## *log1pf()*

## Description

Compute natural logarithm plus one, float.

## Prototype

```
float log1pf(float x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-e logarithm of  $x+1$ .

## *log1pl()*

## Description

Compute natural logarithm plus one, long double.

## Prototype

```
long double log1pl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute logarithm of.

## Return value

- If  $x = \text{NaN}$ , return  $x$ .
- If  $x < 0$ , return  $\text{NaN}$ .
- If  $x = 0$ , return negative infinity.
- If  $x$  is positively infinite, return infinity.
- Else, return base-e logarithm of  $x+1$ .

## [ldexp\(\)](#)

## Description

Scale by power of two, double.

## Prototype

```
double ldexp(double x,  
             int  n);
```

## Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

## Return value

- If  $x$  is  $\pm 0$ , return  $x$ ;
- If  $x$  is  $\pm\infty$ , return  $x$ .
- If  $x$  is  $\text{NaN}$ , return  $x$ .
- Else, return  $x * 2^n$ .

## Additional information

Multiplies a floating-point number by an integral power of two.

See also

## [scalbn\(\)](#)



## *ldexpf()*

### Description

Scale by power of two, float.

### Prototype

```
float ldexpf(float x,  
             int  n);
```

### Parameters

Parameter	Description
x	Value to scale.
n	Power of two to scale by.

### Return value

- If x is zero, return x;
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * 2^n$ .

### Additional information

Multiplies a floating-point number by an integral power of two.

See also

## *scalbnf()*

## *ldexpl()*

### Description

Scale by power of two, long double.

### Prototype

```
long double ldexpl(long double x,  
                   int          n);
```

### Parameters

Parameter	Description
x	Value to scale.

n            Power of two to scale by.

Return value

- If x is  $\pm 0$ , return x;
- If x is  $\pm \infty$ , return x.
- If x is NaN, return x.
- Else, return  $x * 2^n$ .

Additional information

Multiplies a floating-point number by an integral power of two.

See also

[scalbn\(\)](#)

[pow\(\)](#)

Description

Raise to power, double.

Prototype

```
double pow(double x,  
           double y);
```

Parameters

Parameter	Description
x	Base.
y	Power.

Return value

Return x raised to the power y.

[powf\(\)](#)

Description

Raise to power, float.

Prototype

```
float powf(float x,  
           float y);
```

## Parameters

Parameter	Description
x	Base.
y	Power.

## Return value

Return x raised to the power y.

*powl()*

## Description

Raise to power, long double.

## Prototype

```
long double powl(long double x,  
                 long double y);
```

## Parameters

Parameter	Description
x	Base.
y	Power.

## Return value

Return x raised to the power y.

*scalbn()*

## Description

Scale, double.

## Prototype

```
double scalbn(double x,  
              int   n);
```

## Parameters

Parameter	Description
x	Value to scale.
n	Power of DBL_RADIX to scale by.

### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * \text{DBL\_RADIX}^n$ .

### Additional information

Multiplies a floating-point number by an integral power of DBL\_RADIX.

As floating-point arithmetic conforms to IEC 60559, DBL\_RADIX is 2 and [scalbn\(\)](#) is (in this implementation) identical to [ldexp\(\)](#).

See also

[ldexp\(\)](#)

[scalbnf\(\)](#)

### Description

Scale, float.

### Prototype

```
float scalbnf(float x,  
              int  n);
```

### Parameters

Parameter	Description
x	Value to scale.
n	Power of FLT_RADIX to scale by.

### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * \text{FLT\_RADIX}^n$ .

### Additional information

Multiplies a floating-point number by an integral power of FLT\_RADIX.

As floating-point arithmetic conforms to IEC 60559, FLT\_RADIX is 2 and [scalbnf\(\)](#) is (in this implementation) identical to [ldexpf\(\)](#).

See also

[ldexpf\(\)](#)

[scalbnl\(\)](#)

Description

Scale, long double.

Prototype

```
long double scalbnl(long double x,  
                    int          n);
```

Parameters

Parameter	Description
x	Value to scale.
n	Power of LDBL_RADIX to scale by.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * \text{LDBL\_RADIX}^n$ .

Additional information

Multiplies a floating-point number by an integral power of LDBL\_RADIX.

As floating-point arithmetic conforms to IEC 60559, LDBL\_RADIX is 2 and [scalbnl\(\)](#) is (in this implementation) identical to [ldexpl\(\)](#).

See also

[ldexpl\(\)](#)

[scalbln\(\)](#)

Description

Scale, double.

Prototype

```
double scalbln(double x,  
               long   n);
```

Parameters

---

Parameter	Description
x	Value to scale.
n	Power of DBL_RADIX to scale by.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * \text{DBL\_RADIX}^n$ .

#### Additional information

Multiplies a floating-point number by an integral power of DBL\_RADIX.

As floating-point arithmetic conforms to IEC 60559, DBL\_RADIX is 2 and [scalbln\(\)](#) is (in this implementation) identical to [ldexp\(\)](#).

See also

[ldexp\(\)](#)

[scalblnf\(\)](#)

#### Description

Scale, float.

#### Prototype

```
float scalblnf(float x,  
               long n);
```

#### Parameters

---

Parameter	Description
x	Value to scale.
n	Power of FLT_RADIX to scale by.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * \text{FLT\_RADIX}^n$ .

#### Additional information

Multiplies a floating-point number by an integral power of FLT\_RADIX.

As floating-point arithmetic conforms to IEC 60559, FLT\_RADIX is 2 and `scalbnf()` is (in this implementation) identical to `ldexpf()`.

### `scalblnl()`

#### Description

Scale, long double.

#### Prototype

```
long double scalblnl(long double x,  
                    long          n);
```

#### Parameters

Parameter	Description
x	Value to scale.
n	Power of LDBL_RADIX to scale by.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return  $x * \text{LDBL\_RADIX}^n$ .

#### Additional information

Multiplies a floating-point number by an integral power of LDBL\_RADIX.

As floating-point arithmetic conforms to IEC 60559, LDBL\_RADIX is 2 and `scalblnl()` is (in this implementation) identical to `ldexpl()`.

See also

### `ldexpl()`

#### Trigonometric functions

Function	Description
<code>sin()</code>	Calculate sine, double.
<code>sinf()</code>	Calculate sine, float.
<code>sinl()</code>	Calculate sine, long double.
<code>cos()</code>	Calculate cosine, double.

---

<code>cosf()</code>	Calculate cosine, float.
<code>cosl()</code>	Calculate cosine, long double.
<code>tan()</code>	Compute tangent, double.
<code>tanf()</code>	Compute tangent, float.
<code>tanl()</code>	Compute tangent, long double.
<code>sinh()</code>	Compute hyperbolic sine, double.
<code>sinhf()</code>	Compute hyperbolic sine, float.
<code>sinhl()</code>	Compute hyperbolic sine, long double.
<code>cosh()</code>	Compute hyperbolic cosine, double.
<code>coshf()</code>	Compute hyperbolic cosine, float.
<code>coshl()</code>	Compute hyperbolic cosine, long double.
<code>tanh()</code>	Compute hyperbolic tangent, double.
<code>tanhf()</code>	Compute hyperbolic tangent, float.
<code>tanhf()</code>	Compute hyperbolic tangent, long double.

### *sin()*

#### Description

Calculate sine, double.

#### Prototype

```
double sin(double x);
```

#### Parameters

Parameter	Description
x	Angle to compute sine of, radians.

#### Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

### *sinf()*

#### Description

Calculate sine, float.

---



## Prototype

```
float sinf(float x);
```

## Parameters

Parameter	Description
x	Angle to compute sine of, radians.

## Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

## *sinl()*

## Description

Calculate sine, long double.

## Prototype

```
long double sinl(long double x);
```

## Parameters

Parameter	Description
x	Angle to compute sine of, radians.

## Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular sine of x.

## *cos()*

## Description

Calculate cosine, double.

## Prototype

```
double cos(double x);
```

## Parameters

---

Parameter	Description
-----------	-------------

x	Angle to compute cosine of, radians.
---	--------------------------------------

#### Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

#### *cosf()*

#### Description

Calculate cosine, float.

#### Prototype

```
float cosf(float x);
```

#### Parameters

---

Parameter	Description
-----------	-------------

x	Angle to compute cosine of, radians.
---	--------------------------------------

#### Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

#### *cosl()*

#### Description

Calculate cosine, long double.

#### Prototype

```
long double cosl(long double x);
```

#### Parameters

---

Parameter	Description
-----------	-------------

x	Angle to compute cosine of, radians.
---	--------------------------------------

#### Return value

- If x is NaN, return x.
- If x is infinite, return NaN.
- Else, return circular cosine of x.

### *tan()*

#### Description

Compute tangent, double.

#### Prototype

```
double tan(double x);
```

#### Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

#### Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.
- Else, return tangent of x.

### *tanf()*

#### Description

Compute tangent, float.

#### Prototype

```
float tanf(float x);
```

#### Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

#### Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.

- Else, return tangent of x.

### *tanl()*

#### Description

Compute tangent, long double.

#### Prototype

```
long double tanl(long double x);
```

#### Parameters

Parameter	Description
x	Angle to compute tangent of, radians.

#### Return value

- If x is zero, return x.
- If x is infinite, return NaN.
- If x is NaN, return x.
- Else, return tangent of x.

### *sinh()*

#### Description

Compute hyperbolic sine, double.

#### Prototype

```
double sinh(double x);
```

#### Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

#### Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

## *sinhf()*

### Description

Compute hyperbolic sine, float.

### Prototype

```
float sinhf(float x);
```

### Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

### Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

## *sinhl()*

### Description

Compute hyperbolic sine, long double.

### Prototype

```
long double sinhl(long double x);
```

### Parameters

Parameter	Description
x	Value to compute hyperbolic sine of.

### Return value

- If x is NaN, return x.
- If x is infinite, return x.
- Else, return hyperbolic sine of x.

## *cosh()*

### Description

Compute hyperbolic cosine, double.

## Prototype

```
double cosh(double x);
```

## Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

## Return value

- If x is NaN, return x.
- If x is infinite, return  $+\infty$ .
- Else, return hyperbolic cosine of x.

## *coshf()*

## Description

Compute hyperbolic cosine, float.

## Prototype

```
float coshf(float x);
```

## Parameters

Parameter	Description
x	Value to compute hyperbolic cosine of.

## Return value

- If x is NaN, return x.
- If x is infinite, return  $+\infty$ .
- Else, return hyperbolic cosine of x.

## *coshl()*

## Description

Compute hyperbolic cosine, long double.

## Prototype

```
long double coshl(long double x);
```

## Parameters

---

Parameter	Description
x	Value to compute hyperbolic cosine of.

Return value

- If x is NaN, return x.
- If x is infinite, return  $+\infty$ .
- Else, return hyperbolic cosine of x.

*tanh()*

Description

Compute hyperbolic tangent, double.

Prototype

```
double tanh(double x);
```

Parameters

---

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.
- Else, return hyperbolic tangent of x.

*tanhf()*

Description

Compute hyperbolic tangent, float.

Prototype

```
float tanhf(float x);
```

Parameters

---

Parameter	Description
x	Value to compute hyperbolic tangent of.

Return value

- If x is NaN, return x.

- Else, return hyperbolic tangent of  $x$ .

*tanh()*

### Description

Compute hyperbolic tangent, long double.

## Prototype

```
long double tanhl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute hyperbolic tangent of.

### Return value

- If  $x$  is NaN, return  $x$ .
- Else, return hyperbolic tangent of  $x$ .

## Inverse trigonometric functions

Function	Description
<code>asin()</code>	Compute inverse sine, double.
<code>asinf()</code>	Compute inverse sine, float.
<code>asinl()</code>	Compute inverse sine, long double.
<code>acos()</code>	Compute inverse cosine, double.
<code>acosf()</code>	Compute inverse cosine, float.
<code>acosl()</code>	Compute inverse cosine, long double.
<code>atan()</code>	Compute inverse tangent, double.
<code>atanf()</code>	Compute inverse tangent, float.
<code>atanl()</code>	Compute inverse tangent, long double.
<code>atan2()</code>	Compute inverse tangent, with quadrant, double.
<code>atan2f()</code>	Compute inverse tangent, with quadrant, float.
<code>atan2l()</code>	Compute inverse tangent, with quadrant, long double.
<code>asinh()</code>	Compute inverse hyperbolic sine, double.
<code>asinhf()</code>	Compute inverse hyperbolic sine, float.
<code>asinhf()</code>	Compute inverse hyperbolic sine, long double.
<code>acosh()</code>	Compute inverse hyperbolic cosine, double.



`acoshf()` Compute inverse hyperbolic cosine, float.  
`acoshl()` Compute inverse hyperbolic cosine, long double.  
`atanh()` Compute inverse hyperbolic tangent, double.  
`atanhf()` Compute inverse hyperbolic tangent, float.  
`atanhl()` Compute inverse hyperbolic tangent, long double.

## `asin()`

### Description

Compute inverse sine, double.

### Prototype

```
double asin(double x);
```

### Parameters

Parameter	Description
x	Value to compute inverse sine of.

### Return value

- If x is NaN, return x.
- If  $|x| > 1$ , return NaN.
- Else, return inverse circular sine of x.

### Additional information

Calculates the principal value, in radians, of the inverse circular sine of x. The principal value lies in the interval  $[-\pi/2, \pi/2]$  radians.

## `asinf()`

### Description

Compute inverse sine, float.

### Prototype

```
float asinf(float x);
```

### Parameters

Parameter	Description
x	Value to compute inverse sine of.

#### Return value

- If  $x$  is NaN, return  $x$ .
- If  $|x| > 1$ , return NaN.
- Else, return inverse circular sine of  $x$ .

#### Additional information

Calculates the principal value, in radians, of the inverse circular sine of  $x$ . The principal value lies in the interval  $[-\pi/2, \pi/2]$  radians.

#### *asinl()*

#### Description

Compute inverse sine, long double.

#### Prototype

```
long double asinl(long double x);
```

#### Parameters

Parameter	Description
$x$	Value to compute inverse sine of.

#### Return value

- If  $x$  is NaN, return  $x$ .
- If  $|x| > 1$ , return NaN.
- Else, return inverse circular sine of  $x$ .

#### Additional information

Calculates the principal value, in radians, of the inverse circular sine of  $x$ . The principal value lies in the interval  $[-\pi/2, \pi/2]$  radians.

#### *acos()*

#### Description

Compute inverse cosine, double.

#### Prototype

```
double acos(double x);
```

#### Parameters

---

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If  $|x| > 1$ , return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval  $[0, \text{Pi}]$  radians.

*acosf()*

Description

Compute inverse cosine, float.

Prototype

```
float acosf(float x);
```

Parameters

---

Parameter	Description
x	Value to compute inverse cosine of.

Return value

- If x is NaN, return x.
- If  $|x| > 1$ , return NaN.
- Else, return inverse circular cosine of x.

Additional information

Calculates the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval  $[0, \text{Pi}]$  radians.

*acosl()*

Description

Compute inverse cosine, long double.

Prototype

---

---

```
long double acosl(long double x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse cosine of.

#### Return value

- If x is NaN, return x.
- If  $|x| > 1$ , return NaN.
- Else, return inverse circular cosine of x.

#### Additional information

Calculates the principal value, in radians, of the inverse circular cosine of x. The principal value lies in the interval  $[0, \text{Pi}]$  radians.

#### *atan()*

#### Description

Compute inverse tangent, double.

#### Prototype

```
double atan(double x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse tangent of.

#### Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

#### Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval  $[-\text{Pi}/2, \text{Pi}/2]$  radians.

#### *atanf()*

#### Description

Compute inverse tangent, float.

---

## Prototype

```
float atanf(float x);
```

## Parameters

Parameter	Description
x	Value to compute inverse tangent of.

## Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

## Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval  $[-\pi/2, \pi/2]$  radians.

## [atanl\(\)](#)

## Description

Compute inverse tangent, long double.

## Prototype

```
long double atanl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute inverse tangent of.

## Return value

- If x is NaN, return x.
- Else, return inverse tangent of x.

## Additional information

Calculates the principal value, in radians, of the inverse tangent of x. The principal value lies in the interval  $[-\pi/2, \pi/2]$  radians.

## [atan2\(\)](#)

## Description

Compute inverse tangent, with quadrant, double.

Prototype

```
double atan2(double y,  
             double x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval  $[-\pi, +\pi]$  radians.

[\*atan2f\(\)\*](#)

Description

Compute inverse tangent, with quadrant, float.

Prototype

```
float atan2f(float y,  
             float x);
```

Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

Return value

Inverse tangent of y/x.

Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval  $[-\pi, +\pi]$  radians.

### *atan2l()*

#### Description

Compute inverse tangent, with quadrant, long double.

#### Prototype

```
long double atan2l(long double y,  
                  long double x);
```

#### Parameters

Parameter	Description
y	Rise value of angle.
x	Run value of angle.

#### Return value

Inverse tangent of y/x.

#### Additional information

This calculates the value, in radians, of the inverse tangent of y divided by x using the signs of x and y to compute the quadrant of the return value. The principal value lies in the interval  $[-\pi, +\pi]$  radians.

### *asinh()*

#### Description

Compute inverse hyperbolic sine, double.

#### Prototype

```
double asinh(double x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

### *asinhf()*

#### Description

Compute inverse hyperbolic sine, float.

#### Prototype

```
float asinhf(float x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return inverse hyperbolic sine of x.

#### Additional information

Calculates the inverse hyperbolic sine of x.

### *asinh1()*

#### Description

Compute inverse hyperbolic sine, long double.

#### Prototype

```
long double asinh1(long double x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic sine of.

#### Return value

- If x is infinite, return x.



- If  $x$  is NaN, return  $x$ .
- Else, return inverse hyperbolic sine of  $x$ .

### *acosh()*

#### Description

Compute inverse hyperbolic cosine, double.

#### Prototype

```
double acosh(double x);
```

#### Parameters

Parameter	Description
$x$	Value to compute inverse hyperbolic cosine of.

#### Return value

- If  $x < 1$ , return NaN.
- If  $x$  is NaN, return  $x$ .
- Else, return non-negative inverse hyperbolic cosine of  $x$ .

### *acoshf()*

#### Description

Compute inverse hyperbolic cosine, float.

#### Prototype

```
float acoshf(float x);
```

#### Parameters

Parameter	Description
$x$	Value to compute inverse hyperbolic cosine of.

#### Return value

- If  $x < 1$ , return NaN.
- If  $x$  is NaN, return  $x$ .
- Else, return non-negative inverse hyperbolic cosine of  $x$ .

### *acoshl()*

#### Description

Compute inverse hyperbolic cosine, long double.

#### Prototype

```
long double acoshl(long double x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic cosine of.

#### Return value

- If  $x < 1$ , return NaN.
- If  $x$  is NaN, return  $x$ .
- Else, return non-negative inverse hyperbolic cosine of  $x$ .

### *atanh()*

#### Description

Compute inverse hyperbolic tangent, double.

#### Prototype

```
double atanh(double x);
```

#### Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

#### Return value

- If  $x$  is NaN, return  $x$ .
- If  $|x| > 1$ , return NaN.
- If  $x = +/-1$ , return  $+/-$ infinity.
- Else, return non-negative inverse hyperbolic tangent of  $x$ .

### *atanhf()*

#### Description

Compute inverse hyperbolic tangent, float.

Prototype

```
float atanhf(float x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If  $|x| > 1$ , return NaN.
- If  $x = +/-1$ , return  $+/-$ infinity.
- Else, return non-negative inverse hyperbolic tangent of x.

[\*atanhl\(\)\*](#)

Description

Compute inverse hyperbolic tangent, long double.

Prototype

```
long double atanh1(long double x);
```

Parameters

Parameter	Description
x	Value to compute inverse hyperbolic tangent of.

Return value

- If x is NaN, return x.
- If  $|x| > 1$ , return NaN.
- If  $x = +/-1$ , return  $+/-$ infinity.
- Else, return non-negative inverse hyperbolic tangent of x.

### Special functions

Function	Description
<a href="#"><i>erf()</i></a>	Error function, double.
<a href="#"><i>erff()</i></a>	Error function, float.
<a href="#"><i>erfl()</i></a>	Error function, long double.

<code>erfc()</code>	Complementary error function, double.
<code>erfcf()</code>	Complementary error function, float.
<code>erfcl()</code>	Complementary error function, long double.
<code>lgamma()</code>	Log-Gamma function, double.
<code>lgammaf()</code>	Log-Gamma function, float.
<code>lgammal()</code>	Log-Gamma function, long double.
<code>tgamma()</code>	Gamma function, double.
<code>tgammaf()</code>	Gamma function, float.
<code>tgammal()</code>	Gamma function, long double.

### `erf()`

#### Description

Error function, double.

#### Prototype

```
double erf(double x);
```

#### Parameters

Parameter	Description
<code>x</code>	Argument.

#### Return value

`erf(x)`.

### `erff()`

#### Description

Error function, float.

#### Prototype

```
float erff(float x);
```

#### Parameters

Parameter	Description
<code>x</code>	Argument.

#### Return value

`erf(x)`.

[\*erfl\(\)\*](#)

Description

Error function, long double.

Prototype

```
long double erfl(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

`erf(x)`.

[\*erfc\(\)\*](#)

Description

Complementary error function, double.

Prototype

```
double erfc(double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

`erfc(x)`.

[\*erfcf\(\)\*](#)

Description

Complementary error function, float.

Prototype

```
float erfcf(float x);
```

## Parameters

Parameter	Description
x	Argument.

## Return value

erfc(x).

*erfcl()*

## Description

Complementary error function, long double.

## Prototype

```
long double erfcl(long double x);
```

## Parameters

Parameter	Description
x	Argument.

## Return value

erfc(x).

*lgamma()*

## Description

Log-Gamma function, double.

## Prototype

```
double lgamma(double x);
```

## Parameters

Parameter	Description
x	Argument.

## Return value

log(gamma(x)).

### *lgammaf()*

Description

Log-Gamma function, float.

Prototype

```
float lgammaf(float x);
```

Parameters

Parameter	Description
x	Argument.

Return value

$\log(\text{gamma}(x))$ .

### *lgammal()*

Description

Log-Gamma function, long double.

Prototype

```
long double lgammal(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

$\log(\text{gamma}(x))$ .

### *tgamma()*

Description

Gamma function, double.

Prototype

```
double tgamma(double x);
```

Parameters

---

Parameter	Description
x	Argument.

Return value

gamma(x).

*tgammaf()*

Description

Gamma function, float.

Prototype

```
float tgammaf(float x);
```

Parameters

Parameter	Description
x	Argument.

Return value

gamma(x).

*tgammal()*

Description

Gamma function, long double.

Prototype

```
long double tgammal(long double x);
```

Parameters

Parameter	Description
x	Argument.

Return value

gamma(x).

### Rounding and remainder functions

---

Function	Description
----------	-------------

---



---

<code>ceil()</code>	Compute smallest integer not less than, double.
<code>ceilf()</code>	Compute smallest integer not less than, float.
<code>ceill()</code>	Compute smallest integer not less than, long double.
<code>floor()</code>	Compute largest integer not greater than, double.
<code>floorf()</code>	Compute largest integer not greater than, float.
<code>floorl()</code>	Compute largest integer not greater than, long double.
<code>trunc()</code>	Truncate to integer, double.
<code>truncf()</code>	Truncate to integer, float.
<code>truncl()</code>	Truncate to integer, long double.
<code>rint()</code>	Round to nearest integer, double.
<code>rintf()</code>	Round to nearest integer, float.
<code>rintl()</code>	Round to nearest integer, long double.
<code>lrint()</code>	Round to nearest integer, double.
<code>lrintf()</code>	Round to nearest integer, float.
<code>lrintl()</code>	Round to nearest integer, long double.
<code>llrint()</code>	Round to nearest integer, double.
<code>llrintf()</code>	Round to nearest integer, float.
<code>llrintl()</code>	Round to nearest integer, long double.
<code>round()</code>	Round to nearest integer, double.
<code>roundf()</code>	Round to nearest integer, float.
<code>roundl()</code>	Round to nearest integer, long double.
<code>lround()</code>	Round to nearest integer, double.
<code>lroundf()</code>	Round to nearest integer, float.
<code>lroundl()</code>	Round to nearest integer, long double.
<code>llround()</code>	Round to nearest integer, double.
<code>llroundf()</code>	Round to nearest integer, float.
<code>llroundl()</code>	Round to nearest integer, long double.
<code>nearbyint()</code>	Round to nearest integer, double.
<code>nearbyintf()</code>	Round to nearest integer, float.
<code>nearbyintl()</code>	Round to nearest integer, long double.
<code>fmod()</code>	Compute remainder after division, double.
<code>fmodf()</code>	Compute remainder after division, float.
<code>fmodl()</code>	Compute remainder after division, long double.

---

---

<code>modf()</code>	Separate integer and fractional parts, double.
<code>modff()</code>	Separate integer and fractional parts, float.
<code>modfl()</code>	Separate integer and fractional parts, long double.
<code>remainder()</code>	Compute remainder after division, double.
<code>remainderf()</code>	Compute remainder after division, float.
<code>remainderl()</code>	Compute remainder after division, long double.
<code>remquo()</code>	Compute remainder after division, double.
<code>remquof()</code>	Compute remainder after division, float.
<code>remquol()</code>	Compute remainder after division, long double.

### *ceil()*

#### Description

Compute smallest integer not less than, double.

#### Prototype

```
double ceil(double x);
```

#### Parameters

Parameter	Description
x	Value to compute ceiling of.

#### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

### *ceilf()*

#### Description

Compute smallest integer not less than, float.

#### Prototype

```
float ceilf(float x);
```

#### Parameters

---

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

*ceil()*

Description

Compute smallest integer not less than, long double.

Prototype

```
long double ceil(long double x);
```

Parameters

---

Parameter	Description
x	Value to compute ceiling of.

Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the smallest integer value not greater than x.

*floor()*

Description

Compute largest integer not greater than, double.

Prototype

```
double floor(double x);
```

Parameters

---

Parameter	Description
x	Value to floor.

### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

### *floorf()*

### Description

Compute largest integer not greater than, float.

### Prototype

```
float floorf(float x);
```

### Parameters

Parameter	Description
x	Value to floor.

### Return value

- If x is zero, return x.
- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

### *floorl()*

### Description

Compute largest integer not greater than, long double.

### Prototype

```
long double floorl(long double x);
```

### Parameters

Parameter	Description
x	Value to floor.

### Return value

- If x is zero, return x.

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the largest integer value not greater than x.

### *trunc()*

#### Description

Truncate to integer, double.

#### Prototype

```
double trunc(double x);
```

#### Parameters

Parameter	Description
x	Value to truncate.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x with fractional part removed.

### *truncf()*

#### Description

Truncate to integer, float.

#### Prototype

```
float truncf(float x);
```

#### Parameters

Parameter	Description
x	Value to truncate.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x with fractional part removed.

### *trunc1()*

#### Description

Truncate to integer, long double.

#### Prototype

```
long double trunc1(long double x);
```

#### Parameters

Parameter	Description
x	Value to truncate.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return x with fractional part removed.

### *rint()*

#### Description

Round to nearest integer, double.

#### Prototype

```
double rint(double x);
```

#### Parameters

Parameter	Description
x	Value to compute nearest integer of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

### *rintf()*

#### Description

Round to nearest integer, float.

## Prototype

```
float rintf(float x);
```

## Parameters

Parameter	Description
x	Value to compute nearest integer of.

## Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *rintl()*

## Description

Round to nearest integer, long double.

## Prototype

```
long double rintl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute nearest integer of.

## Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *lrint()*

## Description

Round to nearest integer, double.

## Prototype

```
long lrint(double x);
```

## Parameters

---

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

*lrintf()*

Description

Round to nearest integer, float.

Prototype

```
long lrintf(float x);
```

Parameters

---

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

*lrintl()*

Description

Round to nearest integer, long double.

Prototype

```
long lrintl(long double x);
```

Parameters

---

Parameter	Description
x	Value to compute nearest integer of.

Return value



- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

### *llrint()*

#### Description

Round to nearest integer, double.

#### Prototype

```
long long llrint(double x);
```

#### Parameters

Parameter	Description
x	Value to compute nearest integer of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

### *llrintf()*

#### Description

Round to nearest integer, float.

#### Prototype

```
long long llrintf(float x);
```

#### Parameters

Parameter	Description
x	Value to compute nearest integer of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *llrintl()*

### Description

Round to nearest integer, long double.

### Prototype

```
long long llrintl(long double x);
```

### Parameters

Parameter	Description
x	Value to compute nearest integer of.

### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *round()*

### Description

Round to nearest integer, double.

### Prototype

```
double round(double x);
```

### Parameters

Parameter	Description
x	Value to compute nearest integer of.

### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x, ties away from zero.

## *roundf()*

### Description

Round to nearest integer, float.

## Prototype

```
float roundf(float x);
```

## Parameters

Parameter	Description
x	Value to compute nearest integer of.

## Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x, ties away from zero.

## *roundl()*

## Description

Round to nearest integer, long double.

## Prototype

```
long double roundl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute nearest integer of.

## Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x, ties away from zero.

## *lround()*

## Description

Round to nearest integer, double.

## Prototype

```
long lround(double x);
```

## Parameters

---

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

*lroundf()*

Description

Round to nearest integer, float.

Prototype

```
long lroundf(float x);
```

Parameters

---

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

*lroundl()*

Description

Round to nearest integer, long double.

Prototype

```
long lroundl(long double x);
```

Parameters

---

Parameter	Description
x	Value to compute nearest integer of.

Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

### *llround()*

#### Description

Round to nearest integer, double.

#### Prototype

```
long long llround(double x);
```

#### Parameters

Parameter	Description
x	Value to compute nearest integer of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

### *llroundf()*

#### Description

Round to nearest integer, float.

#### Prototype

```
long long llroundf(float x);
```

#### Parameters

Parameter	Description
x	Value to compute nearest integer of.

#### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *llroundl()*

### Description

Round to nearest integer, long double.

### Prototype

```
long long llroundl(long double x);
```

### Parameters

Parameter	Description
x	Value to compute nearest integer of.

### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *nearbyint()*

### Description

Round to nearest integer, double.

### Prototype

```
double nearbyint(double x);
```

### Parameters

Parameter	Description
x	Value to compute nearest integer of.

### Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *nearbyintf()*

### Description

Round to nearest integer, float.

## Prototype

```
float nearbyintf(float x);
```

## Parameters

Parameter	Description
x	Value to compute nearest integer of.

## Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *nearbyintl()*

## Description

Round to nearest integer, long double.

## Prototype

```
long double nearbyintl(long double x);
```

## Parameters

Parameter	Description
x	Value to compute nearest integer of.

## Return value

- If x is infinite, return x.
- If x is NaN, return x.
- Else, return the nearest integer value to x.

## *fmod()*

## Description

Compute remainder after division, double.

## Prototype

```
double fmod(double x,  
            double y);
```

## Parameters

---

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### *fmodf()*

#### Description

Compute remainder after division, float.

#### Prototype

```
float fmodf(float x,  
            float y);
```

#### Parameters

---

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.



- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### *fmodl()*

##### Description

Compute remainder after division, long double.

##### Prototype

```
long double fmodl(long double x,  
                  long double y);
```

##### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

##### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### *modf()*

##### Description

Separate integer and fractional parts, double.

##### Prototype

```
double modf(double x,  
            double * iptr);
```

#### Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

#### Return value

The signed fractional part of x.

#### Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and `modf()` returns the signed fractional part of x.

#### `modff()`

#### Description

Separate integer and fractional parts, float.

#### Prototype

```
float modff(float x,  
            float * iptr);
```

#### Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

#### Return value

The signed fractional part of x.

#### Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and `modff()` returns the signed fractional part of x.

## *modfl()*

### Description

Separate integer and fractional parts, long double.

### Prototype

```
long double modfl(long double x,  
                  long double * iptr);
```

### Parameters

Parameter	Description
x	Value to separate.
iptr	Pointer to object that receives the integral part of x.

### Return value

The signed fractional part of x.

### Additional information

Breaks x into integral and fractional parts, each of which has the same type and sign as x.

The integral part (in floating-point format) is stored in the object pointed to by iptr and [modf\(\)](#) returns the signed fractional part of x.

## *remainder()*

### Description

Compute remainder after division, double.

### Prototype

```
double remainder(double x,  
                  double y);
```

### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

### Return value

- If x is NaN, return NaN.

- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### *remainderf()*

#### Description

Compute remainder after division, float.

#### Prototype

```
float remainderf(float x,  
                float y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

## *remainderl()*

### Description

Compute remainder after division, long double.

### Prototype

```
long double remainderl(long double x,  
                        long double y);
```

### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

## *remquo()*

### Description

Compute remainder after division, double.

### Prototype

```
double remquo(double x,  
              double y,  
              int * quo);
```

### Parameters

Parameter	Description
-----------	-------------

x            Value #1.  
y            Value #2.  
quo         Pointer to object that receives the integer part of x divided by y.

#### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### *remquof()*

#### Description

Compute remainder after division, float.

#### Prototype

```
float remquof(float x,  
              float y,  
              int * quo);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.
quo	Pointer to object that receives the integer part of x divided by y.

#### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.

- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### *remquo()*

#### Description

Compute remainder after division, long double.

#### Prototype

```
long double remquo1(long double x,  
                    long double y,  
                    int * quo);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.
quo	Pointer to object that receives the integer part of x divided by y.

#### Return value

- If x is NaN, return NaN.
- If x is zero and y is nonzero, return x.
- If x is infinite, return NaN.
- If x is finite and y is infinite, return x.
- If y is NaN, return NaN.
- If y is zero, return NaN.
- Else, return remainder of x divided by y.

#### Additional information

Computes the floating-point remainder of x divided by y, i.e. the value  $x - i*y$  for some integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.

#### Absolute value functions

Function	Description
----------	-------------

`fabs()`      Compute absolute value, double.  
`fabsf()`    Compute absolute value, float.  
`fabsl()`    Compute absolute value, long double.

### *fabs()*

#### Description

Compute absolute value, double.

#### Prototype

```
double fabs(double x);
```

#### Parameters

Parameter	Description
x	Value to compute magnitude of.

#### Return value

- If x is NaN, return x.
- Else, absolute value of x.

### *fabsf()*

#### Description

Compute absolute value, float.

#### Prototype

```
float fabsf(float x);
```

#### Parameters

Parameter	Description
x	Value to compute magnitude of.

#### Return value

- If x is NaN, return x.
- Else, absolute value of x.

### *fabsl()*

#### Description



Compute absolute value, long double.

Prototype

```
long double fabsl(long double x);
```

Parameters

Parameter	Description
x	Value to compute magnitude of.

Return value

- If x is NaN, return x.
- Else, absolute value of x.

### Fused multiply functions

Function	Description
<a href="#">fma()</a>	Compute fused multiply-add, double.
<a href="#">fmaf()</a>	Compute fused multiply-add, float.
<a href="#">fmal()</a>	Compute fused multiply-add, long double.

#### [fma\(\)](#)

Description

Compute fused multiply-add, double.

Prototype

```
double fma(double x,  
           double y,  
           double z);
```

Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.
z	Summand.

Return value

Return  $(x * y) + z$ .

## *fmaf()*

### Description

Compute fused multiply-add, float.

### Prototype

```
float fmaf(float x,  
           float y,  
           float z);
```

### Parameters

Parameter	Description
x	Multiplier.
y	Multiplicand.
z	Summand.

### Return value

Return  $(x * y) + z$ .

## *fmal()*

### Description

Compute fused multiply-add, long double.

### Prototype

```
long double fmal(long double x,  
                 long double y,  
                 long double z);
```

### Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.
z	Summand.

### Return value

Return  $(x * y) + z$ .

## Maximum, minimum, and positive difference functions

Function	Description
<a href="#">fmin()</a>	Compute minimum, double.
<a href="#">fminf()</a>	Compute minimum, float.
<a href="#">fminl()</a>	Compute minimum, long double.
<a href="#">fmax()</a>	Compute maximum, double.
<a href="#">fmaxf()</a>	Compute maximum, float.
<a href="#">fmaxl()</a>	Compute maximum, long double.
<a href="#">fdim()</a>	Positive difference, double.
<a href="#">fdimf()</a>	Positive difference, float.
<a href="#">fdiml()</a>	Positive difference, long double.

### [fmin\(\)](#)

#### Description

Compute minimum, double.

#### Prototype

```
double fmin(double x,  
            double y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

### [fminf\(\)](#)

#### Description

Compute minimum, float.

#### Prototype

```
float fminf(float x,  
            float y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

#### *fminl()*

#### Description

Compute minimum, long double.

#### Prototype

```
long double fminl(long double x,  
                  long double y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return minimum of x and y.

#### *fmax()*

#### Description

Compute maximum, double.

#### Prototype

```
double fmax(double x,  
            double y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

#### *fmaxf()*

#### Description

Compute maximum, float.

#### Prototype

```
float fmaxf(float x,  
            float y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

#### *fmaxl()*

#### Description

Compute maximum, long double.

#### Prototype

```
long double fmaxl(long double x,  
                  long double y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If x is NaN, return y.
- If y is NaN, return x.
- Else, return maximum of x and y.

#### *fdim()*

#### Description

Positive difference, double.

#### Prototype

```
double fdim(double x,  
            double y);
```

#### Parameters

Parameter	Description
x	Value #1.
y	Value #2.

#### Return value

- If  $x > y$ ,  $x - y$ .
- Else,  $+0$ .

#### *fdimf()*

#### Description

Positive difference, float.

#### Prototype

```
float fdimf(float x,  
            float y);
```

## Parameters

Parameter	Description
x	Value #1.
y	Value #2.

## Return value

- If  $x > y$ ,  $x-y$ .
- Else,  $+0$ .

## *fdiml()*

## Description

Positive difference, long double.

## Prototype

```
long double fdiml(long double x,  
                  long double y);
```

## Parameters

Parameter	Description
x	Value #1.
y	Value #2.

## Return value

- If  $x > y$ ,  $x-y$ .
- Else,  $+0$ .

## Miscellaneous functions

Function	Description
<a href="#">nextafter()</a>	Next machine-floating value, double.
<a href="#">nextafterf()</a>	Next machine-floating value, float.
<a href="#">nextafterl()</a>	Next machine-floating value, long double.
<a href="#">nexttoward()</a>	Next machine-floating value, double.
<a href="#">nexttowardf()</a>	Next machine-floating value, float.
<a href="#">nexttowardl()</a>	Next machine-floating value, long double.
<a href="#">nan()</a>	Parse NaN, double.
<a href="#">nanf()</a>	Parse NaN, float.

`nanl()` Parse NaN, long double.  
`copysign()` Copy sign, double.  
`copysignf()` Copy sign, float.  
`copysignl()` Copy sign, long double.

### *nextafter()*

#### Description

Next machine-floating value, double.

#### Prototype

```
double nextafter(double x,  
                 double y);
```

#### Parameters

Parameter	Description
x	Value to step from.
y	Director to step in.

#### Return value

Next machine-floating value after x in direction of y.

### *nextafterf()*

#### Description

Next machine-floating value, float.

#### Prototype

```
float nextafterf(float x,  
                 float y);
```

#### Parameters

Parameter	Description
x	Value to step from.
y	Director to step in.

#### Return value

Next machine-floating value after x in direction of y.



### *nextafterl()*

#### Description

Next machine-floating value, long double.

#### Prototype

```
long double nextafterl(long double x,  
                      long double y);
```

#### Parameters

Parameter	Description
x	Value to step from.
y	Director to step in.

#### Return value

Next machine-floating value after x in direction of y.

### *nexttoward()*

#### Description

Next machine-floating value, double.

#### Prototype

```
double nexttoward(double x,  
                  long double y);
```

#### Parameters

Parameter	Description
x	Value to step from.
y	Direction to step in.

#### Return value

Next machine-floating value after x in direction of y.

### *nexttowardf()*

#### Description

Next machine-floating value, float.

## Prototype

```
float nexttowardf(float x,  
                  long double y);
```

## Parameters

Parameter	Description
x	Value to step from.
y	Direction to step in.

## Return value

Next machine-floating value after x in direction of y.

[\*nexttowardl\(\)\*](#)

## Description

Next machine-floating value, long double.

## Prototype

```
long double nexttowardl(long double x,  
                        long double y);
```

## Parameters

Parameter	Description
x	Value to step from.
y	Direction to step in.

## Return value

Next machine-floating value after x in direction of y.

[\*nan\(\)\*](#)

## Description

Parse NaN, double.

## Prototype

```
double nan(const char * tag);
```

## Parameters

---

Parameter	Description
-----------	-------------

tag	NaN tag.
-----	----------

Return value

Quiet NaN formed from tag.

*nanf()*

Description

Parse NaN, float.

Prototype

```
float nanf(const char * tag);
```

Parameters

---

Parameter	Description
-----------	-------------

tag	NaN tag.
-----	----------

Return value

Quiet NaN formed from tag.

*nanl()*

Description

Parse NaN, long double.

Prototype

```
long double nanl(const char * tag);
```

Parameters

---

Parameter	Description
-----------	-------------

tag	NaN tag.
-----	----------

Return value

Quiet NaN formed from tag.

*copysign()*

Description

---

Copy sign, double.

Prototype

```
double copysign(double x,  
                double y);
```

Parameters

Parameter	Description
x	Floating value to inject sign into.
y	Floating value carrying the sign to inject.

Return value

x with the sign of y.

[\*copysignf\(\)\*](#)

Description

Copy sign, float.

Prototype

```
float copysignf(float x,  
                float y);
```

Parameters

Parameter	Description
x	Floating value to inject sign into.
y	Floating value carrying the sign to inject.

Return value

x with the sign of y.

[\*copysignl\(\)\*](#)

Description

Copy sign, long double.

Prototype

```
long double copysignl(long double x,  
                      long double y);
```

## Parameters

Parameter	Description
x	Floating value to inject sign into.
y	Floating value carrying the sign to inject.

## Return value

x with the sign of y.

## <setjmp.h>

### Non-local flow control

#### *setjmp()*

## Description

Save calling environment for non-local jump.

## Prototype

```
int setjmp(jmp_buf buf);
```

## Parameters

Parameter	Description
buf	Buffer to save context into.

## Return value

On return from a direct invocation, returns the value zero. On return from a call to the [longjmp\(\)](#) function, returns a nonzero value determined by the call to [longjmp\(\)](#).

## Additional information

Saves its calling environment in env for later use by the [longjmp\(\)](#) function.

The environment saved by a call to [setjmp\(\)](#) consists of information sufficient for a call to the [longjmp\(\)](#) function to return execution to the correct block and invocation of that block, were it called recursively.

#### *longjmp()*

## Description

Restores the saved environment.

## Prototype

```
void longjmp(jmp_buf buf,  
            int    val);
```

## Parameters

Parameter	Description
buf	Buffer to restore context from.
val	Value to return to <code>setjmp()</code> call.

## Additional information

Restores the environment saved by `setjmp()` in the corresponding env argument. If there has been no such invocation, or if the function containing the invocation of `setjmp()` has terminated execution in the interim, the behavior of `longjmp()` is undefined.

After `longjmp()` is completed, program execution continues as if the corresponding invocation of `setjmp()` had just returned the value specified by val.

Objects of automatic storage allocation that are local to the function containing the invocation of the corresponding `setjmp()` that do not have volatile-qualified type and have been changed between the `setjmp()` invocation and `longjmp()` call are indeterminate.

## Notes

`longjmp()` cannot cause `setjmp()` to return the value 0; if val is 0, `setjmp()` returns the value 1.

## <stdbool.h>

### Macros

#### *bool*

## Description

Macros expanding to support the Boolean type.

## Definition

```
#define bool    _Bool  
#define true    1  
#define false   0
```

## Symbols

Definition	Description
------------	-------------

---

bool	Underlying boolean type
true	Boolean true value
false	Boolean false value

## <stddef.h>

### Macros

#### NULL

##### Description

Null-pointer constant.

##### Definition

```
#define NULL    0
```

##### Symbols

Definition	Description
NULL	Null pointer

#### offsetof

##### Description

Calculate offset of member from start of structure.

##### Definition

```
#define offsetof(s,m)    ((size_t)&(((s *)0)->m))
```

##### Symbols

Definition	Description
offsetof(s,m)	Offset of m within s

### Types

#### size\_t

##### Description

Unsigned integral type returned by the sizeof operator.

##### Type definition

```
typedef __SEGGER_RTL_SIZE_T size_t;
```

[ptrdiff\\_t](#)

Description

Signed integral type of the result of subtracting two pointers.

Type definition

```
typedef __SEGGER_RTL_PTRDIFF_T ptrdiff_t;
```

[wchar\\_t](#)

Description

Integral type that can hold one wide character.

Type definition

```
typedef __SEGGER_RTL_WCHAR_T wchar_t;
```

**<stdint.h>**

**Minima and maxima**

*Signed integer minima and maxima*

Description

Minimum and maximum values for signed integer types.

Definition

```
#define INT8_MIN      (-128)
#define INT8_MAX      127
#define INT16_MIN     (-32767-1)
#define INT16_MAX     32767
#define INT32_MIN     (-2147483647L-1)
#define INT32_MAX     2147483647L
#define INT64_MIN     (-9223372036854775807LL-1)
#define INT64_MAX     9223372036854775807LL
```

Symbols

Definition	Description
INT8_MIN	Minimum value of <a href="#">int8_t</a>
INT8_MAX	Maximum value of <a href="#">int8_t</a>



INT16\_MIN    Minimum value of [int16\\_t](#)  
INT16\_MAX    Maximum value of [int16\\_t](#)  
INT32\_MIN    Minimum value of [int32\\_t](#)  
INT32\_MAX    Maximum value of [int32\\_t](#)  
INT64\_MIN    Minimum value of [int64\\_t](#)  
INT64\_MAX    Maximum value of [int64\\_t](#)

### *Unsigned integer minima and maxima*

#### Description

Minimum and maximum values for unsigned integer types.

#### Definition

```
#define UINT8_MAX      255  
#define UINT16_MAX     65535  
#define UINT32_MAX     4294967295UL  
#define UINT64_MAX     18446744073709551615ULL
```

#### Symbols

Definition	Description
UINT8_MAX	Maximum value of <a href="#">uint8_t</a>
UINT16_MAX	Maximum value of <a href="#">uint16_t</a>
UINT32_MAX	Maximum value of <a href="#">uint32_t</a>
UINT64_MAX	Maximum value of <a href="#">uint64_t</a>

### *Maximal integer minima and maxima*

#### Description

Minimum and maximum values for signed and unsigned maximal-integer types.

#### Definition

```
#define INTMAX_MIN      INT64_MIN  
#define INTMAX_MAX      INT64_MAX  
#define UINTMAX_MAX     UINT64_MAX
```

#### Symbols

Definition	Description
INTMAX_MIN	Minimum value of <a href="#">intmax_t</a>

INTMAX\_MAX     Maximum value of [intmax\\_t](#)  
 UINTMAX\_MAX    Maximum value of [uintmax\\_t](#)

### *Least integer minima and maxima*

#### Description

Minimum and maximum values for signed and unsigned least-integer types.

#### Definition

```
#define INT_LEAST8_MIN     INT8_MIN
#define INT_LEAST8_MAX     INT8_MAX
#define INT_LEAST16_MIN    INT16_MIN
#define INT_LEAST16_MAX    INT16_MAX
#define INT_LEAST32_MIN    INT32_MIN
#define INT_LEAST32_MAX    INT32_MAX
#define INT_LEAST64_MIN    INT64_MIN
#define INT_LEAST64_MAX    INT64_MAX
#define UINT_LEAST8_MAX    UINT8_MAX
#define UINT_LEAST16_MAX   UINT16_MAX
#define UINT_LEAST32_MAX   UINT32_MAX
#define UINT_LEAST64_MAX   UINT64_MAX
```

#### Symbols

Definition	Description
INT_LEAST8_MIN	Minimum value of <a href="#">int_least8_t</a>
INT_LEAST8_MAX	Maximum value of <a href="#">int_least8_t</a>
INT_LEAST16_MIN	Minimum value of <a href="#">int_least16_t</a>
INT_LEAST16_MAX	Maximum value of <a href="#">int_least16_t</a>
INT_LEAST32_MIN	Minimum value of <a href="#">int_least32_t</a>
INT_LEAST32_MAX	Maximum value of <a href="#">int_least32_t</a>
INT_LEAST64_MIN	Minimum value of <a href="#">int_least64_t</a>
INT_LEAST64_MAX	Maximum value of <a href="#">int_least64_t</a>
UINT_LEAST8_MAX	Maximum value of <a href="#">uint_least8_t</a>
UINT_LEAST16_MAX	Maximum value of <a href="#">uint_least16_t</a>
UINT_LEAST32_MAX	Maximum value of <a href="#">uint_least32_t</a>
UINT_LEAST64_MAX	Maximum value of <a href="#">uint_least64_t</a>

## Fast integer minima and maxima

### Description

Minimum and maximum values for signed and unsigned fast-integer types.

### Definition

```
#define INT_FAST8_MIN      INT8_MIN
#define INT_FAST8_MAX      INT8_MAX
#define INT_FAST16_MIN     INT32_MIN
#define INT_FAST16_MAX     INT32_MAX
#define INT_FAST32_MIN     INT32_MIN
#define INT_FAST32_MAX     INT32_MAX
#define INT_FAST64_MIN     INT64_MIN
#define INT_FAST64_MAX     INT64_MAX
#define UINT_FAST8_MAX     UINT8_MAX
#define UINT_FAST16_MAX    UINT32_MAX
#define UINT_FAST32_MAX    UINT32_MAX
#define UINT_FAST64_MAX    UINT64_MAX
```

### Symbols

Definition	Description
INT_FAST8_MIN	Minimum value of <a href="#">int_fast8_t</a>
INT_FAST8_MAX	Maximum value of <a href="#">int_fast8_t</a>
INT_FAST16_MIN	Minimum value of <a href="#">int_fast16_t</a>
INT_FAST16_MAX	Maximum value of <a href="#">int_fast16_t</a>
INT_FAST32_MIN	Minimum value of <a href="#">int_fast32_t</a>
INT_FAST32_MAX	Maximum value of <a href="#">int_fast32_t</a>
INT_FAST64_MIN	Minimum value of <a href="#">int_fast64_t</a>
INT_FAST64_MAX	Maximum value of <a href="#">int_fast64_t</a>
UINT_FAST8_MAX	Maximum value of <a href="#">uint_fast8_t</a>
UINT_FAST16_MAX	Maximum value of <a href="#">uint_fast16_t</a>
UINT_FAST32_MAX	Maximum value of <a href="#">uint_fast32_t</a>
UINT_FAST64_MAX	Maximum value of <a href="#">uint_fast64_t</a>

## Pointer types minima and maxima

### Description

Minimum and maximum values for pointer-related types.

## Definition

```
#define PTRDIFF_MIN    INT64_MIN
#define PTRDIFF_MAX    INT64_MAX
#define SIZE_MAX       INT64_MAX
#define INTPTR_MIN     INT64_MIN
#define INTPTR_MAX     INT64_MAX
#define UINTPTR_MAX    UINT64_MAX
```

## Symbols

Definition	Description
PTRDIFF_MIN	Minimum value of <a href="#">ptrdiff_t</a>
PTRDIFF_MAX	Maximum value of <a href="#">ptrdiff_t</a>
SIZE_MAX	Maximum value of <a href="#">size_t</a>
INTPTR_MIN	Minimum value of <a href="#">intptr_t</a>
INTPTR_MAX	Maximum value of <a href="#">intptr_t</a>
UINTPTR_MAX	Maximum value of <a href="#">uintptr_t</a>
PTRDIFF_MIN	Minimum value of <a href="#">ptrdiff_t</a>
PTRDIFF_MAX	Maximum value of <a href="#">ptrdiff_t</a>
SIZE_MAX	Maximum value of <a href="#">size_t</a>
INTPTR_MIN	Minimum value of <a href="#">intptr_t</a>
INTPTR_MAX	Maximum value of <a href="#">intptr_t</a>
UINTPTR_MAX	Maximum value of <a href="#">uintptr_t</a>

## *Wide integer minima and maxima*

## Description

Minimum and maximum values for the `wint_t` type.

## Definition

```
#define WINT_MIN    (-2147483647L-1)
#define WINT_MAX    2147483647L
```

## Symbols

Definition	Description
WINT_MIN	Minimum value of <code>wint_t</code>
WINT_MAX	Maximum value of <code>wint_t</code>

## Constant construction macros

### *Signed integer construction macros*

#### Description

Macros that create constants of type `intx_t`.

#### Definition

```
#define INT8_C(x)      (x)
#define INT16_C(x)     (x)
#define INT32_C(x)     (x)
#define INT64_C(x)     (x##LL)
```

#### Symbols

Definition	Description
<code>INT8_C(x)</code>	Create constant of type <code>int8_t</code>
<code>INT16_C(x)</code>	Create constant of type <code>int16_t</code>
<code>INT32_C(x)</code>	Create constant of type <code>int32_t</code>
<code>INT64_C(x)</code>	Create constant of type <code>int64_t</code>

### *Unsigned integer construction macros*

#### Description

Macros that create constants of type `uintx_t`.

#### Definition

```
#define UINT8_C(x)     (x##u)
#define UINT16_C(x)    (x##u)
#define UINT32_C(x)    (x##u)
#define UINT64_C(x)    (x##uLL)
```

#### Symbols

Definition	Description
<code>UINT8_C(x)</code>	Create constant of type <code>uint8_t</code>
<code>UINT16_C(x)</code>	Create constant of type <code>uint16_t</code>
<code>UINT32_C(x)</code>	Create constant of type <code>uint32_t</code>
<code>UINT64_C(x)</code>	Create constant of type <code>uint64_t</code>

## Maximal integer construction macros

### Description

Macros that create constants of type `intmax_t` and `uintmax_t`.

### Definition

```
#define INTMAX_C(x)      (x##LL)
#define UINTMAX_C(x)    (x##uLL)
```

### Symbols

Definition	Description
<code>INTMAX_C(x)</code>	Create constant of type <code>intmax_t</code>
<code>UINTMAX_C(x)</code>	Create constant of type <code>uintmax_t</code>

## <stdio.h>

### Formatted output control strings

The functions in this section that accept a formatted output control string do so according to the specification that follows.

### Composition

The format is composed of zero or more directives: ordinary characters (not %, which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments, converting them, if applicable, according to the corresponding conversion specifier, and then writing the result to the output stream.

Each conversion specification is introduced by the character %. After the % the following appear in sequence:

- Zero or more flags (in any order) that modify the meaning of the conversion specification.
- An optional minimum field width. If the converted value has fewer characters than the field width, it is padded with spaces (by default) on the left (or right, if the left adjustment flag has been given) to the field width. The field width takes the form of an asterisk \* or a decimal integer.
- An optional precision that gives the minimum number of digits to appear for the d, i, o, u, x, and X conversions, the number of digits to appear after the decimal-point character for e, E, f, and F conversions, the maximum number of significant digits for the g and G conversions, or the maximum number of bytes to be written for s conversions. The precision takes the form of a period . followed either by an asterisk \* or by an optional decimal integer; if only the period is

specified, the precision is taken as zero. If a precision appears with any other conversion specifier, the behavior is undefined.

- An optional length modifier that specifies the size of the argument.
- A conversion specifier character that specifies the type of conversion to be applied.

As noted above, a field width, or precision, or both, may be indicated by an asterisk. In this case, an int argument supplies the field width or precision. The arguments specifying field width, or precision, or both, must appear (in that order) before the argument (if any) to be converted. A negative field width argument is taken as a - flag followed by a positive field width. A negative precision argument is taken as if the precision were omitted.

### *Flag characters*

The flag characters and their meanings are:

Flag	Description
-	The result of the conversion is left-justified within the field. The default, if this flag is not specified, is that the result of the conversion is left-justified within the field.
+	The result of a signed conversion always begins with a plus or minus sign. The default, if this flag is not specified, is that it begins with a sign only when a negative value is converted.
space	If the first character of a signed conversion is not a sign, or if a signed conversion results in no characters, a space is prefixed to the result. If the space and + flags both appear, the space flag is ignored.
#	The result is converted to an alternative form. For o conversion, it increases the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both zero, a single 0 is printed). For x or X conversion, a nonzero result has 0x or 0X prefixed to it. For e, E, f, F, g, and G conversions, the result of converting a floating-point number always contains a decimal-point character, even if no digits follow it. (Normally, a decimal-point character appears in the result of these conversions only if a digit follows it.) For g and G conversions, trailing zeros are not removed from the result. As an extension, when used in p conversion, the results has # prefixed to it. For other conversions, the behavior is undefined.
0	For d, i, o, u, x, X, e, E, f, F, g, and G conversions, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the 0 and - flags both appear, the 0 flag is ignored. For d, i, o, u, x, and X conversions, if a precision is specified, the 0 flag is ignored. For other conversions, the behavior is undefined.

### *Length modifiers*

The length modifiers and their meanings are:

Flag	Description
hh	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a signed char or unsigned char argument (the argument will have been promoted according to the integer promotions, but its value will be converted to signed char or unsigned char before printing); or that a following n conversion specifier applies to a pointer to a signed char argument.
h	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a short int or unsigned short int argument (the argument will have been promoted according to the integer promotions, but its value is converted to short int or unsigned short int before printing); or that a following n conversion specifier applies to a pointer to a short int argument.
l	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long int or unsigned long int argument; that a following n conversion specifier applies to a pointer to a long int argument; or has no effect on a following e, E, f, F, g, or G conversion specifier.
ll	Specifies that a following d, i, o, u, x, or X conversion specifier applies to a long long int or unsigned long long int argument; that a following n conversion specifier applies to a pointer to a long long int argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

### Conversion specifiers

The conversion specifiers and their meanings are:

Flag	Description
d, i	The argument is converted to signed decimal in the style [-]dddd. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
o, u, x, X	The unsigned argument is converted to unsigned octal for o, unsigned decimal for u, or unsigned hexadecimal notation for x or X in the style dddd the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it is expanded with leading spaces. The default precision is one. The result of converting a zero value with a precision of zero is no characters.
f, F	A double argument representing a floating-point number is converted to decimal notation in the style [-]ddd.ddd, where the number of digits after the decimal-point character is equal to the precision specification. If the precision is missing, it is taken as 6; if the precision is zero and the # flag is not specified, no decimal-point character appears. If a decimal-point character appears, at least one digit appears before it. The value is rounded



to the appropriate number of digits. A double argument representing an infinity is converted to `inf`. A double argument representing a NaN is converted to `nan`. The `F` conversion specifier produces `INF` or `NAN` instead of `inf` or `nan`, respectively.

- e, E A double argument representing a floating-point number is converted in the style `[-]d.ddde±dd`, where there is one digit (which is nonzero if the argument is nonzero) before the decimal-point character and the number of digits after it is equal to the precision; if the precision is missing, it is taken as 6; if the precision is zero and the `#` flag is not specified, no decimal-point character appears. The value is rounded to the appropriate number of digits. The `E` conversion specifier produces a number with `E` instead of `e` introducing the exponent. The exponent always contains at least two digits, and only as many more digits as necessary to represent the exponent. If the value is zero, the exponent is zero. A double argument representing an infinity is converted to `inf`. A double argument representing a NaN is converted to `nan`. The `E` conversion specifier produces `INF` or `NAN` instead of `inf` or `nan`, respectively.
- g, G A double argument representing a floating-point number is converted in style `f` or `e` (or in style `F` or `E` in the case of a `G` conversion specifier), with the precision specifying the number of significant digits. If the precision is zero, it is taken as one. The style used depends on the value converted; style `e` (or `E`) is used only if the exponent resulting from such a conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the fractional portion of the result unless the `#` flag is specified; a decimal-point character appears only if it is followed by a digit. A double argument representing an infinity is converted to `inf`. A double argument representing a NaN is converted to `nan`. The `G` conversion specifier produces `INF` or `NAN` instead of `inf` or `nan`, respectively.
- c The argument is converted to an unsigned char, and the resulting character is written.
- s The argument is be a pointer to the initial element of an array of character type. Characters from the array are written up to (but not including) the terminating null character. If the precision is specified, no more than that many characters are written. If the precision is not specified or is greater than the size of the array, the array must contain a null character.
- p The argument is a pointer to void. The value of the pointer is converted in the same format as the `x` conversion specifier with a fixed precision of `2*sizeof(void *)`.
- n The argument is a pointer to a signed integer into which is written the number of characters written to the output stream so far by the call to the formatting function. No argument is converted, but one is consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.
- % A `%` character is written. No argument is converted.

Note that the C99 width modifier `l` used in conjunction with the `c` and `s` conversion specifiers is not supported and nor are the conversion specifiers `a` and `A`.

## Formatted input control strings

The format is composed of zero or more directives: one or more white-space characters, an ordinary character (neither % nor a white-space character), or a conversion specification.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- An optional assignment-suppressing character \*.
- An optional nonzero decimal integer that specifies the maximum field width (in characters).
- An optional length modifier that specifies the size of the receiving object.
- A conversion specifier character that specifies the type of conversion to be applied.

The formatted input function executes each directive of the format in turn. If a directive fails, the function returns. Failures are described as input failures (because of the occurrence of an encoding error or the unavailability of input characters), or matching failures (because of inappropriate input).

A directive composed of white-space character(s) is executed by reading input up to the first non-white-space character (which remains unread), or until no more characters can be read.

A directive that is an ordinary character is executed by reading the next characters of the stream. If any of those characters differ from the ones composing the directive, the directive fails and the differing and subsequent characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from being read, the directive fails.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each specifier. A conversion specification is executed in the following steps:

- Input white-space characters (as specified by the `isspace()` function) are skipped, unless the specification includes a `[`, `c`, or `n` specifier.
- An input item is read from the stream, unless the specification includes an `n` specifier. An input item is defined as the longest sequence of input characters which does not exceed any specified field width and which is, or is a prefix of, a matching input sequence. The first character, if any, after the input item remains unread. If the length of the input item is zero, the execution of the directive fails; this condition is a matching failure unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.
- Except in the case of a % specifier, the input item (or, in the case of a %n directive, the count of input characters) is converted to a type appropriate to the conversion specifier. If the input item is not a matching sequence, the execution of the directive fails: this condition is a matching failure. Unless assignment suppression was indicated by a \*, the result of the conversion is placed in the object pointed to by the first argument following the format argument that has not already received a conversion result. If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the object, the behavior is undefined.

### Length modifiers

The length modifiers and their meanings are:

Flag	Description
hh	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to signed char or pointer to unsigned char.
h	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to short int or unsigned short int.
l	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long int or unsigned long int; that a following e, E, f, F, g, or G conversion specifier applies to an argument with type pointer to double.
ll	Specifies that a following d, i, o, u, x, X, or n conversion specifier applies to an argument with type pointer to long long int or unsigned long long int.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined. Note that the C99 length modifiers j, z, and t are not supported.

### Conversion specifiers

Flag	Description
d	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the <code>strtol()</code> function with the value 10 for the base argument. The corresponding argument must be a pointer to signed integer.
i	Matches an optionally signed integer, whose format is the same as expected for the subject sequence of the <code>strtol()</code> function with the value zero for the base argument. The corresponding argument must be a pointer to signed integer.
o	Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of the <code>strtol()</code> function with the value 18 for the base argument. The corresponding argument must be a pointer to signed integer.
u	Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of the <code>strtoul()</code> function with the value 10 for the base argument. The corresponding argument must be a pointer to unsigned integer.
x	Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of the <code>strtoul()</code> function with the value 16 for the base argument. The corresponding argument must be a pointer to unsigned integer.
e, f, g	Matches an optionally signed floating-point number whose format is the same as expected for the subject sequence of the <code>strtod()</code> function. The corresponding argument shall be a pointer to floating.
c	Matches a sequence of characters of exactly the number specified by the field width (one if no field width is present in the directive). The corresponding argument must be a pointer

to the initial element of a character array large enough to accept the sequence. No null character is added.

- s Matches a sequence of non-white-space characters. The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence and a terminating null character, which will be added automatically.
- [ Matches a nonempty sequence of characters from a set of expected characters (the scanset). The corresponding argument must be a pointer to the initial element of a character array large enough to accept the sequence and a terminating null character, which will be added automatically. The conversion specifier includes all subsequent characters in the format string, up to and including the matching right bracket `]`. The characters between the brackets (the scanlist) compose the scanset, unless the character after the left bracket is a circumflex `^`, in which case the scanset contains all characters that do not appear in the scanlist between the circumflex and the right bracket. If the conversion specifier begins with `[]` or `^[^]`, the right bracket character is in the scanlist and the next following right bracket character is the matching right bracket that ends the specification; otherwise the first following right bracket character is the one that ends the specification. If a `-` character is in the scanlist and is not the first, nor the second where the first character is a `^`, nor the last character, it is treated as a member of the scanset.
- p Reads a sequence output by the corresponding `%p` formatted output conversion. The corresponding argument must be a pointer to a pointer to void.
- n No input is consumed. The corresponding argument shall be a pointer to signed integer into which is to be written the number of characters read from the input stream so far by this call to the formatted input function. Execution of a `%n` directive does not increment the assignment count returned at the completion of execution of the `fscanf` function. No argument is converted, but one is consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.
- % Matches a single `%` character; no conversion or assignment occurs.

Note that the C99 width modifier `l` used in conjunction with the `c`, `s`, and `[` conversion specifiers is not supported and nor are the conversion specifiers `a` and `A`.

### Character and string I/O functions

Function	Description
<code>getchar()</code>	Read character from standard input.
<code>gets()</code>	Read string from standard input.
<code>putc()</code>	Write character to file.
<code>putchar()</code>	Write character to standard output.
<code>puts()</code>	Write string to standard output.

## *getchar()*

### Description

Read character from standard input.

### Prototype

```
int getchar(void);
```

### Return value

If the stream is at end-of-file or a read error occurs, returns EOF, otherwise a nonnegative value.

### Additional information

Reads a single character from the standard input stream.

## *gets()*

### Description

Read string from standard input.

### Prototype

```
char *gets(char * s);
```

### Parameters

Parameter	Description
s	Pointer to object that receives the string.

### Return value

Returns s if successful. If end-of-file is encountered and no characters have been read into the array, the contents of the array remain unchanged and a null pointer is returned. If a read error occurs during the operation, the array contents are indeterminate and a null pointer is returned.

### Additional information

This function reads characters from standard input into the array pointed to by s until end-of-file is encountered or a new-line character is read. Any new-line character is discarded, and a null character is written immediately after the last character read into the array.

## *putc()*

### Description

Write character to file.

Prototype

```
int putc(int    c,  
         FILE * stream);
```

Parameters

Parameter	Description
c	Character to write.
stream	Pointer to stream to write to.

Return value

If no error, the character written. If a write error occurs, returns EOF.

Additional information

Writes the character c to stream.

*putchar()*

Description

Write character to standard output.

Prototype

```
int putchar(int c);
```

Parameters

Parameter	Description
c	Character to write.

Return value

If no error, the character written. If a write error occurs, returns EOF.

Additional information

Writes the character c to the standard output stream.

*puts()*

Description

Write string to standard output.

## Prototype

```
int puts(const char * s);
```

## Parameters

Parameter	Description
s	Pointer to zero-terminated string.

## Return value

Returns EOF if a write error occurs; otherwise it returns a nonnegative value.

## Additional information

Writes the string pointed to by s to the standard output stream using [putchar\(\)](#) and appends a new-line character to the output. The terminating null character is not written.

## Formatted input functions

Function	Description
<a href="#">scanf()</a>	Formatted read from standard input.
<a href="#">sscanf()</a>	Formatted read from string.
<a href="#">vscanf()</a>	Formatted read from standard input, variadic.
<a href="#">vsscanf()</a>	Formatted read from string, variadic.

### [scanf\(\)](#)

## Description

Formatted read from standard input.

## Prototype

```
int scanf(const char * format,  
          ...);
```

## Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.

## Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

## Additional information

Reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

### *sscanf()*

#### Description

Formatted read from string.

#### Prototype

```
int sscanf(const char * s,  
           const char * format,  
           ...);
```

#### Parameters

Parameter	Description
s	Pointer to string to read from.
format	Pointer to zero-terminated format control string.

#### Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

## Additional information

Reads input from the string s under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

### *vscanf()*

#### Description

Formatted read from standard input, variadic.



## Prototype

```
int vscanf(const char    * format,
           va_list      arg);
```

## Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

## Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

## Additional information

Reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. Before calling [vscanf\(\)](#), arg must be initialized by the va\_start() macro (and possibly subsequent va\_arg() calls). [vscanf\(\)](#) does not invoke the va\_end() macro.

If there are insufficient arguments for the format, the behavior is undefined.

## [vsscanf\(\)](#)

## Description

Formatted read from string, variadic.

## Prototype

```
int vsscanf(const char    * s,
            const char    * format,
            va_list      arg);
```

## Parameters

Parameter	Description
s	Pointer to string to read from.
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

## Return value

Returns the value of the macro EOF if an input failure occurs before any conversion. Otherwise, returns the number of input items assigned, which can be fewer than provided for, or even zero, in the event of an early matching failure.

## Additional information

Reads input from the standard input stream under control of the string pointed to by format that specifies the admissible input sequences and how they are to be converted for assignment, using subsequent arguments as pointers to the objects to receive the converted input. Before calling `vsscanf()`, arg must be initialized by the `va_start()` macro (and possibly subsequent `va_arg()` calls). `vsscanf()` does not invoke the `va_end()` macro.

If there are insufficient arguments for the format, the behavior is undefined.

## Formatted output functions

Function	Description
<code>printf()</code>	Formatted write to standard output.
<code>sprintf()</code>	Formatted write to string.
<code>snprintf()</code>	Formatted write to string, limit length.
<code>vprintf()</code>	Formatted write to standard output, variadic.
<code>vsprintf()</code>	Formatted write to string, variadic.
<code>vsnprintf()</code>	Formatted write to string, limit length, variadic.

### `printf()`

## Description

Formatted write to standard output.

## Prototype

```
int printf(const char * format,  
          ...);
```

## Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.

## Return value

Returns the number of characters written, or a negative value if an output or encoding error occurred.

## Additional information

Writes to the standard output stream under control of the string pointed to by format that specifies how subsequent arguments are converted for output.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

### *sprintf()*

#### Description

Formatted write to string.

#### Prototype

```
int sprintf(    char * s,  
               const char * format,  
               ...);
```

#### Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
format	Pointer to zero-terminated format control string.

#### Return value

Returns number of characters written to s (not counting the terminating null), or a negative value if an output or encoding error occurred.

## Additional information

Writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output. A null character is written at the end of the characters written; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

### *snprintf()*

#### Description

Formatted write to string, limit length.

## Prototype

```
int snprintf(    char    * s,  
                size_t  n,  
                const char * format,  
                ...);
```

## Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
n	Maximum number of characters to write to the array pointed to by s.
format	Pointer to zero-terminated format control string.

## Return value

Returns the number of characters that would have been written had n been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than n.

## Additional information

Writes to the string pointed to by s under control of the string pointed to by format that specifies how subsequent arguments are converted for output.

If n is zero, nothing is written, and s can be a null pointer. Otherwise, output characters beyond count n-1 are discarded rather than being written to the array, and a null character is written at the end of the characters actually written into the array. A null character is written at the end of the conversion; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

## *vprintf()*

## Description

Formatted write to standard output, variadic.

## Prototype

```
int vprintf(const char * format,  
            va_list  arg);
```

## Parameters

Parameter	Description
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

## Return value

Returns the number of characters written, or a negative value if an output or encoding error occurred.

## Additional information

Writes to the standard output stream using under control of the string pointed to by format that specifies how subsequent arguments are converted for output. Before calling `vprintf()`, arg must be initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). `vprintf()` does not invoke the `va_end` macro.

## `vprintf()`

## Description

Formatted write to string, variadic.

## Prototype

```
int vprintf(      char    * s,  
                const char * format,  
                va_list  arg);
```

## Parameters

Parameter	Description
s	Pointer to array that receives the formatted output.
format	Pointer to zero-terminated format control string.
arg	Variable parameter list.

## Return value

Returns number of characters written to s (not counting the terminating null), or a negative value if an output or encoding error occurred.

## Additional information

Writes to the string pointed to by *s* under control of the string pointed to by *format* that specifies how subsequent arguments are converted for output. A null character is written at the end of the characters written; it is not counted as part of the returned value.

Before calling `vsprintf()`, *arg* must be initialized by the `va_start` macro (and possibly subsequent `va_arg` calls). `vsprintf()` does not invoke the `va_end` macro.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

#### Notes

This is equivalent to `sprintf()` with the variable argument list replaced by *arg*.

#### `vsprintf()`

#### Description

Formatted write to string, limit length, variadic.

#### Prototype

```
int vsnprintf(    char    * s,
                  size_t   n,
                  const char * format,
                  va_list   arg);
```

#### Parameters

Parameter	Description
<i>s</i>	Pointer to array that receives the formatted output.
<i>n</i>	Maximum number of characters to write to the array pointed to by <i>s</i> .
<i>format</i>	Pointer to zero-terminated format control string.
<i>arg</i>	Variable parameter list.

#### Return value

Returns the number of characters that would have been written had *n* been sufficiently large, not counting the terminating null character, or a negative value if an encoding error occurred. Thus, the null-terminated output has been completely written if and only if the returned value is nonnegative and less than *n*.

#### Additional information

Writes to the string pointed to by *s* under control of the string pointed to by *format* that specifies how subsequent arguments are converted for output. Before calling `vsnprintf()`, *arg* must be initialized by the `va_start` macro (and possibly subsequent `va_arg()` calls). `vsnprintf()` does not invoke the `va_end` macro.

If *n* is zero, nothing is written, and *s* can be a null pointer. Otherwise, output characters beyond count *n*-1 are discarded rather than being written to the array, and a null character is written at the end of the characters actually written into the array. A null character is written at the end of the conversion; it is not counted as part of the returned value.

If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

If copying takes place between objects that overlap, the behavior is undefined.

#### Notes

This is equivalent to `snprintf()` with the variable argument list replaced by *arg*.

## <stdlib.h>

### Process control functions

Function	Description
<code>atexit()</code>	Set function to be called on exit.
<code>abort()</code>	Abort execution.

#### `atexit()`

#### Description

Set function to be called on exit.

#### Prototype

```
int atexit(__SEGGER_RTL_exit_func fn);
```

#### Parameters

Parameter	Description
<i>fn</i>	Function to register.

#### Return value

- = 0 Success registering function.
- ≠ 0 Did not register function.

## Additional information

Registers function `fn` to be called when the application has exited. The functions registered with `atexit()` are executed in reverse order of their registration.

### `abort()`

#### Description

Abort execution.

#### Prototype

```
void abort(void);
```

#### Additional information

Calls `exit()` with the exit status -1.

## Integer arithmetic functions

Function	Description
<code>abs()</code>	Calculate absolute value, int.
<code>labs()</code>	Calculate absolute value, long.
<code>llabs()</code>	Calculate absolute value, long long.
<code>div()</code>	Divide returning quotient and remainder, int.
<code>ldiv()</code>	Divide returning quotient and remainder, long.
<code>lldiv()</code>	Divide returning quotient and remainder, long long.

### `abs()`

#### Description

Calculate absolute value, int.

#### Prototype

```
int abs(int Value);
```

#### Parameters

Parameter	Description
Value	Integer value.

#### Return value

The absolute value of the integer argument Value.



### *labs()*

#### Description

Calculate absolute value, long.

#### Prototype

```
long int labs(long int Value);
```

#### Parameters

Parameter	Description
Value	Long integer value.

#### Return value

The absolute value of the long integer argument Value.

### *llabs()*

#### Description

Calculate absolute value, long long.

#### Prototype

```
long long int llabs(long long int Value);
```

#### Parameters

Parameter	Description
Value	Long long integer value.

#### Return value

The absolute value of the long long integer argument Value.

### *div()*

#### Description

Divide returning quotient and remainder, int.

#### Prototype

```
div_t div(int Numer,  
          int Denom);
```

## Parameters

Parameter	Description
Numer	Numerator.
Denom	Demoninator.

## Return value

Returns a structure of type `div_t` comprising both the quotient and the remainder. The structures contain the members `quot` (the quotient) and `rem` (the remainder), each of which has the same type as the arguments `Numer` and `Denom`. If either part of the result cannot be represented, the behavior is undefined.

## Additional information

This computes `Numer` divided by `Denom` and `Numer` modulo `Denom` in a single operation.

See also

`div_t`

[\*ldiv\(\)\*](#)

## Description

Divide returning quotient and remainder, long.

## Prototype

```
ldiv_t ldiv(long Numer,  
            long Denom);
```

## Parameters

Parameter	Description
Numer	Numerator.
Denom	Demoninator.

## Return value

Returns a structure of type `ldiv_t` comprising both the quotient and the remainder. The structures contain the members `quot` (the quotient) and `rem` (the remainder), each of which has the same type as the arguments `Numer` and `Denom`. If either part of the result cannot be represented, the behavior is undefined.

## Additional information

This computes Numer divided by Denom and Numer modulo Denom in a single operation.

See also

ldiv\_t

[\*lldiv\(\)\*](#)

Description

Divide returning quotient and remainder, long long.

Prototype

```
lldiv_t lldiv(long long Numer,  
              long long Denom);
```

Parameters

Parameter	Description
Numer	Numerator.
Denom	Demoninator.

Return value

Returns a structure of type lldiv\_t comprising both the quotient and the remainder. The structures contain the members quot (the quotient) and rem (the remainder), each of which has the same type as the arguments Numer and Denom. If either part of the result cannot be represented, the behavior is undefined.

Additional information

This computes Numer divided by Denom and Numer modulo Denom in a single operation.

See also

lldiv\_t

### Pseudo-random sequence generation functions

Function	Description
<a href="#"><i>rand()</i></a>	Return next random number in sequence.
<a href="#"><i>srand()</i></a>	Set seed of random number sequence.

[\*rand\(\)\*](#)

Description

Return next random number in sequence.

Prototype

```
int rand(void);
```

Return value

Returns the computed pseudo-random integer.

Additional information

This computes a sequence of pseudo-random integers in the range 0 to RAND\_MAX.

See also

[srand\(\)](#)

[srand\(\)](#)

Description

Set seed of random number sequence.

Prototype

```
void srand(unsigned s);
```

Parameters

Parameter	Description
s	New seed value for pseudo-random sequence.

Additional information

This uses the argument Seed as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to [rand\(\)](#). If [srand\(\)](#) is called with the same seed value, the same sequence of pseudo-random numbers is generated.

If [rand\(\)](#) is called before any calls to [srand\(\)](#) have been made, a sequence is generated as if [srand\(\)](#) is first called with a seed value of 1.

See also

[rand\(\)](#)

## Memory allocation functions

Function	Description
----------	-------------

`malloc()` Allocate space for single object.  
`calloc()` Allocate space for multiple objects and zero them.  
`realloc()` Resize or allocate memory space.  
`free()` Free allocated memory for reuse.

### *malloc()*

#### Description

Allocate space for single object.

#### Prototype

```
void *malloc(size_t sz);
```

#### Parameters

Parameter	Description
sz	Number of characters to allocate for the object.

#### Return value

Returns a null pointer if the space for the object cannot be allocated from free memory; if space for the object can be allocated, `malloc()` returns a pointer to the start of the allocated space.

#### Additional information

Allocates space for an object whose size is specified by sz and whose value is indeterminate.

### *calloc()*

#### Description

Allocate space for multiple objects and zero them.

#### Prototype

```
void *calloc(size_t nobj,  
             size_t sz);
```

#### Parameters

Parameter	Description
nobj	Number of objects to allocate.
sz	Number of characters to allocate per object.

#### Return value

Returns a null pointer if the space for the object cannot be allocated from free memory; if space for the object can be allocated, `malloc()` returns a pointer to the start of the allocated space.

#### Additional information

Allocates space for an array of `nobj` objects, each of whose size is `sz`. The space is initialized to all zero bits.

#### `realloc()`

##### Description

Resize or allocate memory space.

##### Prototype

```
void *realloc(void * ptr,  
              size_t  sz);
```

##### Parameters

Parameter	Description
<code>ptr</code>	Pointer to resize, or NULL to allocate.
<code>sz</code>	New size of object.

##### Return value

Returns a pointer to the new object (which may have the same value as a pointer to the old object), or a null pointer if the new object could not be allocated.

#### Additional information

Deallocates the old object pointed to by `ptr` and returns a pointer to a new object that has the size specified by `sz`. The contents of the new object is identical to that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values.

If `ptr` is a null pointer, `realloc()` behaves like `malloc()` for the specified size. If memory for the new object cannot be allocated, the old object is not deallocated and its value is unchanged.

If `ptr` does not match a pointer earlier returned by `calloc()`, `malloc()`, or `realloc()`, or if the space has been deallocated by a call to `free()` or `realloc()`, the behavior is undefined.

#### `free()`

##### Description

Free allocated memory for reuse.

## Prototype

```
void free(void * ptr);
```

## Parameters

Parameter	Description
ptr	Pointer to object to free.

## Additional information

Causes the space pointed to by ptr to be deallocated, that is, made available for further allocation. If ptr is a null pointer, no action occurs.

If ptr does not match a pointer earlier returned by [calloc\(\)](#), [malloc\(\)](#), or [realloc\(\)](#), or if the space has been deallocated by a call to [free\(\)](#) or [realloc\(\)](#), the behavior is undefined.

## Search and sort functions

Function	Description
<a href="#">qsort()</a>	Sort array.
<a href="#">bsearch()</a>	Search sorted array.

### [qsort\(\)](#)

## Description

Sort array.

## Prototype

```
void qsort(void * base,
           size_t nmemb,
           size_t sz,
           int (*compare)(const void * elem1 , const void * elem2 ));
```

## Parameters

Parameter	Description
base	Pointer to the start of the array.
nmemb	Number of array elements.
sz	Number of characters per array element.
compare	Pointer to element comparison function.

## Additional information

Sorts the array pointed to by base using the compare function. The array should have nmemb elements of sz bytes. The compare function should return a negative value if the first parameter is less than the second parameter, zero if the parameters are equal, and a positive value if the first parameter is greater than the second parameter.

### *bsearch()*

#### Description

Search sorted array.

#### Prototype

```
void *bsearch
    (const void * key,
     const void * base,
      size_t    nmemb,
      size_t    sz,
      int       ( *compare)(const void * elem1 , const void * elem2 ));
```

#### Parameters

Parameter	Description
key	Pointer to object to search for.
base	Pointer to the start of the array.
nmemb	Number of array elements.
sz	Number of characters per array element.
compare	Pointer to element comparison function.

#### Return value

= NULL    Key not found.

≠ NULL    Pointer to found object.

#### Additional information

Searches the array pointed to by base for the specified key and returns a pointer to the first entry that matches, or null if no match. The array should have nmemb elements of sz bytes and be sorted by the same algorithm as the compare function.

The compare function should return a negative value if the first parameter is less than second parameter, zero if the parameters are equal, and a positive value if the first parameter is greater than the second parameter.



## Number to string conversions

Function	Description
<a href="#">itoa()</a>	Convert to string, int.
<a href="#">ltoa()</a>	Convert to string, long.
<a href="#">lltoa()</a>	Convert to string, long long.
<a href="#">utoa()</a>	Convert to string, unsigned.
<a href="#">ultoa()</a>	Convert to string, unsigned long.
<a href="#">ulltoa()</a>	Convert to string, unsigned long long.

### *itoa()*

#### Description

Convert to string, int.

#### Prototype

```
char *itoa(int    val,  
           char * buf,  
           int    radix);
```

#### Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

#### Return value

Returns buf.

#### Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

#### Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

[ltoa\(\)](#), [lltoa\(\)](#), [utoa\(\)](#), [ultoa\(\)](#), [ulltoa\(\)](#)

*ltoa()*

Description

Convert to string, long.

Prototype

```
char *ltoa(long    val,  
            char * buf,  
            int    radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

[ltoa\(\)](#), [lltoa\(\)](#), [utoa\(\)](#), [ultoa\(\)](#), [ulltoa\(\)](#)

*lltoa()*

Description

Convert to string, long long.

Prototype

```
char *lltoa(long long    val,  
            char        * buf,  
            int          radix);
```

Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

Return value

Returns buf.

Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

If val is negative and radix is 10, the string has a leading minus sign (-); for all other values of radix, value is considered unsigned and never has a leading minus sign.

Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

[itoa\(\)](#), [ltoa\(\)](#), [utoa\(\)](#), [ultoa\(\)](#), [ulltoa\(\)](#)

[utoa\(\)](#)

Description

Convert to string, unsigned.

Prototype

```
char *utoa(unsigned    val,  
            char        * buf,  
            int          radix);
```

## Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

## Return value

Returns buf.

## Additional information

Converts val to a string in base radix and places the result in buf which must be large enough to hold the output. If radix is greater than 36, the result is undefined.

## Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

[itoa\(\)](#), [ltoa\(\)](#), [lltoa\(\)](#), [ultoa\(\)](#), [ulltoa\(\)](#)

[ultoa\(\)](#)

## Description

Convert to string, unsigned long.

## Prototype

```
char *ultoa(unsigned long    val,  
             char            * buf,  
             int             radix);
```

## Parameters

Parameter	Description
val	Value to convert.
buf	Pointer to array of characters that receives the string.
radix	Number base to use for conversion, 2 to 36.

## Return value

Returns buf.

## Additional information

Converts `val` to a string in base `radix` and places the result in `buf` which must be large enough to hold the output. If `radix` is greater than 36, the result is undefined.

## Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

[itoa\(\)](#), [ltoa\(\)](#), [lltoa\(\)](#), [ulltoa\(\)](#), [utoa\(\)](#)

[ulltoa\(\)](#)

## Description

Convert to string, unsigned long long.

## Prototype

```
char *ulltoa(unsigned long long val,  
             char * buf,  
             int radix);
```

## Parameters

Parameter	Description
<code>val</code>	Value to convert.
<code>buf</code>	Pointer to array of characters that receives the string.
<code>radix</code>	Number base to use for conversion, 2 to 36.

## Return value

Returns `buf`.

## Additional information

Converts `val` to a string in base `radix` and places the result in `buf` which must be large enough to hold the output. If `radix` is greater than 36, the result is undefined.

## Notes

This is a non-standard function. Even though this function is commonly used by compilers on other platforms, there is no guarantee that this function will behave the same on all platforms, in all cases.

See also

[itoa\(\)](#), [ltoa\(\)](#), [lltoa\(\)](#), [ultoa\(\)](#), [utoa\(\)](#)

### String to number conversions

Function	Description
<a href="#">atoi()</a>	Convert to number, int.
<a href="#">atol()</a>	Convert to number, long.
<a href="#">atoll()</a>	Convert to number, long long.
<a href="#">atof()</a>	Convert to number, double.
<a href="#">strtol()</a>	Convert to number, long.
<a href="#">strtoll()</a>	Convert to number, long long.
<a href="#">strtoul()</a>	Convert to number, unsigned long.
<a href="#">strtoull()</a>	Convert to number, unsigned long long.
<a href="#">strtof()</a>	Convert to number, float.
<a href="#">strtod()</a>	Convert to number, double.
<a href="#">strtold()</a>	Convert to number, long double.

[atoi\(\)](#)

#### Description

Convert to number, int.

#### Prototype

```
int atoi(const char * nptr);
```

#### Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.

#### Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

#### Additional information

Converts the initial portion of the string pointed to by `nptr` to an int representation.

[atoi\(\)](#) does not affect the value of `errno` on an error.

#### Notes

Except for the behavior on error, [atoi\(\)](#) is equivalent to `(int)strtol(nptr, NULL, 10)`.

See also

[strtol\(\)](#)

[atol\(\)](#)

Description

Convert to number, long.

Prototype

```
long int atol(const char * nptr);
```

Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.

Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

Additional information

Converts the initial portion of the string pointed to by `nptr` to a long representation.

[atol\(\)](#) does not affect the value of `errno` on an error.

Notes

Except for the behavior on error, [atol\(\)](#) is equivalent to `strtol(nptr, NULL, 10)`.

See also

[strtol\(\)](#)

[atoll\(\)](#)

Description

Convert to number, long long.

Prototype

```
long long int atoll(const char * nptr);
```

## Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.

## Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

## Additional information

Converts the initial portion of the string pointed to by `nptr` to a long-long representation.

[atoll\(\)](#) does not affect the value of `errno` on an error.

## Notes

Except for the behavior on error, [atoll\(\)](#) is equivalent to `strtoll(nptr, NULL, 10)`.

See also

[strtoll\(\)](#)

[atof\(\)](#)

## Description

Convert to number, double.

## Prototype

```
double atof(const char * nptr);
```

## Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.

## Return value

Returns the converted value, if any. If the value of the result cannot be represented, the behavior is undefined.

## Additional information

Converts the initial portion of the string pointed to by `nptr` to an double representation.

[atof\(\)](#) does not affect the value of `errno` on an error.



## Notes

Except for the behavior on error, [atof\(\)](#) is equivalent to `(int)strtod(nptr, NULL)`.

See also

[strtod\(\)](#)

[strtol\(\)](#)

## Description

Convert to number, long.

## Prototype

```
long strtol(const char * nptr,  
            char ** endptr,  
            int base);
```

## Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.
<code>endptr</code>	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
<code>base</code>	Radix to use for conversion, 2 to 36.

## Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LONG_MIN` or `LONG_MAX` is returned according to the sign of the value, if any, and the value of the macro `ERANGE` is stored in `errno`.

## Additional information

Converts the initial portion of the string pointed to by `nptr` to a long representation.

First, [strtol\(\)](#) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by [isspace\(\)](#), a subject sequence resembling an integer represented in some radix determined by the value of `base`, and a final string of one or more unrecognized characters, including the terminating null character of the input string. [strtol\(\)](#) then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as `U`, `L`, `UL`, `LL`, `ULL`) is allowed.

If the value of base is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters “0x” or “0X” may optionally precede the sequence of letters and digits, following the optional sign.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

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 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.

## *strtoll()*

### Description

Convert to number, long long.

### Prototype

```
long long strtoll(const char * nptr,  
                  char ** endptr,  
                  int base);
```

### Parameters

Parameter	Description
nptr	Pointer to string to convert from.

---

endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
base	Radix to use for conversion, 2 to 36.

### Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `LLONG_MIN` or `LLONG_MAX` is returned according to the sign of the value, if any, and the value of the macro `ERANGE` is stored in `errno`.

### Additional information

Converts the initial portion of the string pointed to by `nptr` to a long representation.

First, `strtoll()` decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by `isspace()`, a subject sequence resembling an integer represented in some radix determined by the value of `base`, and a final string of one or more unrecognized characters, including the terminating null character of the input string. `strtoll()` then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as `U`, `L`, `UL`, `LL`, `ULL`) is allowed.

If the value of `base` is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of `base` is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by `base`. The letters from `a` (or `A`) through `z` (or `Z`) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of `base` are permitted.

If the value of `base` is 16, the characters `"0x"` or `"0X"` may optionally precede the sequence of letters and digits, following the optional sign.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of `base` is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of `base` is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

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 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.

## `strtoul()`

### Description

Convert to number, unsigned long.

### Prototype

```
unsigned long strtoul(const char * nptr,  
                    char ** endptr,  
                    int base);
```

### Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.
<code>endptr</code>	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
<code>base</code>	Radix to use for conversion, 2 to 36.

### Return value

`strtoul()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `ULONG_MAX` is and the value of the macro `ERANGE` is stored in `errno`.

### Additional information

Converts the initial portion of the string pointed to by `nptr` to a long int representation.

First, `strtoul()` decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by `isspace()`, a subject sequence resembling an integer represented in some radix determined by the value of `base`, and a final string of one or more unrecognized characters, including the terminating null character of the input string. `strtoul()` then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of `base` is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of base is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by base. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of base are permitted.

If the value of base is 16, the characters “0x” or “0X” may optionally precede the sequence of letters and digits, following the optional sign.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of base is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of base is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

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 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.

## *strtoull()*

### Description

Convert to number, unsigned long long.

### Prototype

```
unsigned long long strtoull(const char * nptr,  
                           char ** endptr,  
                           int base);
```

### Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.
base	Radix to use for conversion, 2 to 36.

## Return value

`strtoull()` returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `ULLONG_MAX` is and the value of the macro `ERANGE` is stored in `errno`.

## Additional information

Converts the initial portion of the string pointed to by `nptr` to a long int representation.

First, `strtoull()` decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by `isspace()`, a subject sequence resembling an integer represented in some radix determined by the value of `base`, and a final string of one or more unrecognized characters, including the terminating null character of the input string. `strtoull()` then attempts to convert the subject sequence to an integer, and return the result.

When converting, no integer suffix (such as U, L, UL, LL, ULL) is allowed.

If the value of `base` is zero, the expected form of the subject sequence is an optional plus or minus sign followed by an integer constant.

If the value of `base` is between 2 and 36 (inclusive), the expected form of the subject sequence is an optional plus or minus sign followed by a sequence of letters and digits representing an integer with the radix specified by `base`. The letters from a (or A) through z (or Z) represent the values 10 through 35; only letters and digits whose ascribed values are less than that of `base` are permitted.

If the value of `base` is 16, the characters “0x” or “0X” may optionally precede the sequence of letters and digits, following the optional sign.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of `base` is zero, the sequence of characters starting with the first digit is interpreted as an integer constant. If the subject sequence has the expected form and the value of `base` is between 2 and 36, it is used as the base for conversion.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated.

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 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.

## strtof()

### Description

Convert to number, float.

### Prototype

```
float strtof(const char * nptr,  
             char ** endptr);
```

### Parameters

Parameter	Description
nptr	Pointer to string to convert from.
endptr	If nonnull, a pointer to object that receives the pointer to the first unconverted character.

### Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, HUGE\_VALF is returned according to the sign of the value, if any, and the value of the macro ERANGE is stored in errno.

### Additional information

Converts the initial portion of the string pointed to by nptr to float representation.

First, [strtof\(\)](#) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by [isspace\(\)](#), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. [strtof\(\)](#) then attempts to convert the subject sequence to a floating-point number, and return the result.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. 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 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.

See also

[strtod\(\)](#)

[strtod\(\)](#)

Description

Convert to number, double.

Prototype

```
double strtod(const char * nptr,  
              char ** endptr);
```

Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.
<code>endptr</code>	If nonnull, a pointer to object that receives the pointer to the first unconverted character.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `HUGE_VAL` is returned according to the sign of the value, if any, and the value of the macro `ERANGE` is stored in `errno`.

Additional information

Converts the initial portion of the string pointed to by `nptr` to double representation.

First, [strtod\(\)](#) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by [isspace\(\)](#), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null character of the input string. [strtod\(\)](#) then attempts to convert the subject sequence to a floating-point number, and return the result.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.



The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. 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 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.

See also

[strtof\(\)](#)

[strtold\(\)](#)

Description

Convert to number, long double.

Prototype

```
long double strtold(const char * nptr,  
                    char ** endptr);
```

Parameters

Parameter	Description
<code>nptr</code>	Pointer to string to convert from.
<code>endptr</code>	If nonnull, a pointer to object that receives the pointer to the first unconverted character.

Return value

Returns the converted value, if any. If no conversion could be performed, zero is returned. If the correct value is outside the range of representable values, `HUGE_VAL` is returned according to the sign of the value, if any, and the value of the macro `ERANGE` is stored in `errno`.

Additional information

Converts the initial portion of the string pointed to by `nptr` to long double representation.

First, [strtold\(\)](#) decomposes the input string into three parts: an initial, possibly empty, sequence of white-space characters, as specified by [isspace\(\)](#), a subject sequence resembling a floating-point constant, and a final string of one or more unrecognized characters, including the terminating null

character of the input string. `strtod()` then attempts to convert the subject sequence to a floating-point number, and return the result.

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 empty or consists entirely of white space, or if the first non-white-space character is other than a sign or a permissible letter or digit.

The expected form of the subject sequence is an optional plus or minus sign followed by a nonempty sequence of decimal digits optionally containing a decimal-point character, then an optional exponent part.

If the subject sequence begins with a minus sign, the value resulting from the conversion is negated. 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 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.

See also

[strtod\(\)](#)

### Multi-byte/wide character functions

Function	Description
<a href="#">btowc()</a>	Convert single-byte character to wide character.
<a href="#">btowc_l()</a>	Convert single-byte character to wide character, per locale, (POSIX.1).
<a href="#">mblen()</a>	Count number of bytes in multi-byte character.
<a href="#">mblen_l()</a>	Count number of bytes in multi-byte character, per locale (POSIX.1).
<a href="#">mbtowc()</a>	Convert multi-byte character to wide character.
<a href="#">mbtowc_l()</a>	Convert multi-byte character to wide character, per locale (POSIX.1).
<a href="#">mbstowcs()</a>	Convert multi-byte string to wide string.
<a href="#">mbstowcs_l()</a>	Convert multi-byte string to wide string, per locale (POSIX.1).
<a href="#">mbsrtowcs()</a>	Convert multi-byte string to wide character string, restartable.
<a href="#">mbsrtowcs_l()</a>	Convert multi-byte string to wide character string, restartable, per locale (POSIX.1).
<a href="#">wctomb()</a>	Convert wide character to multi-byte character.
<a href="#">wctomb_l()</a>	Convert wide character to multi-byte character, per locale (POSIX.1).
<a href="#">wcstombs()</a>	Convert wide string to multi-byte string.
<a href="#">wcstombs_l()</a>	Convert wide string to multi-byte string.

## *btowc()*

### Description

Convert single-byte character to wide character.

### Prototype

```
wint_t btowc(int c);
```

### Parameters

Parameter	Description
c	Character to convert.

### Return value

Returns WEOF if c has the value EOF or if c, converted to an unsigned char and in the current locale, does not constitute a valid single-byte character in the initial shift state.

### Additional information

Determines whether c constitutes a valid single-byte character in the current locale. If c is a valid single-byte character, *btowc()* returns the wide character representation of that character.

## *btowc\_l()*

### Description

Convert single-byte character to wide character, per locale, (POSIX.1).

### Prototype

```
wint_t btowc_l(int c,  
               locale_t loc);
```

### Parameters

Parameter	Description
c	Character to convert.
loc	Locale used for conversion.

### Return value

Returns WEOF if c has the value EOF or if c, converted to an unsigned char and in the locale loc, does not constitute a valid single-byte character in the initial shift state.

### Additional information

Determines whether *c* constitutes a valid single-byte character in the locale *loc*. If *c* is a valid single-byte character, [btowc\\_l\(\)](#) returns the wide character representation of that character.

#### Notes

Conforms to POSIX.1-2008.

#### [mblen\(\)](#)

#### Description

Count number of bytes in multi-byte character.

#### Prototype

```
int mblen(const char * s,
          size_t n);
```

#### Parameters

Parameter	Description
<i>s</i>	Pointer to multi-byte character.
<i>n</i>	Maximum number of bytes to examine.

#### Return value

If *s* is a null pointer, returns a nonzero or zero value, if multi-byte character encodings, respectively, do or do not have state-dependent encodings.

If *s* is not a null pointer, either returns 0 (if *s* points to the null character), or returns the number of bytes that are contained in the multi-byte character (if the next *n* or fewer bytes form a valid multi-byte character), or returns -1 (if they do not form a valid multi-byte character).

#### Additional information

Determines the number of bytes contained in the multi-byte character pointed to by *s* in the current locale.

Except that the conversion state of the [mbtowc\(\)](#) function is not affected, it is equivalent to

```
mbtowc(NULL, s, n);
```

#### See also

[mblen\\_l\(\)](#), [mbtowc\(\)](#)

## [mblen\\_l\(\)](#)

### Description

Count number of bytes in multi-byte character, per locale (POSIX.1).

### Prototype

```
int mblen_l(const char * s,
            size_t n,
            locale_t loc);
```

### Parameters

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.
loc	Locale to use for conversion.

### Return value

If s is a null pointer, returns a nonzero or zero value, if multi-byte character encodings, respectively, do or do not have state-dependent encodings in locale loc.

If s is not a null pointer, either returns 0 (if s points to the null character), or returns the number of bytes that are contained in the multi-byte character (if the next n or fewer bytes form a valid multi-byte character), or returns -1 (if they do not form a valid multi-byte character).

### Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the locale loc.

Except that the conversion state of the [mbtowc\(\)](#) function is not affected, it is equivalent to

```
mbtowc_l(NULL, s, n, loc);
```

### Notes

Conforms to POSIX.1-2008.

See also

[mblen\(\)](#), [mbtowc\(\)](#)

[mbtowc\(\)](#)

### Description

Convert multi-byte character to wide character.

Prototype

```
int mbtowc(      wchar_t * pwc,  
             const char * s,  
             size_t  n);
```

Parameters

Parameter	Description
pwc	Pointer to object that receives the wide character.
s	Pointer to multi-byte character string.
n	Maximum number of bytes that will be examined.

Return value

If s is a null pointer, [mbtowc\(\)](#) returns a nonzero value if multi-byte character encodings are state-dependent in the current locale, and zero otherwise.

If s is not null and the object that s points to is a wide character null, [mbtowc\(\)](#) returns 0.

If s is not null and the object that s points to forms a valid multi-byte character, [mbtowc\(\)](#) returns the length in bytes of the multi-byte character.

If the object that [mbtowc\(\)](#) points to does not form a valid multi-byte character within the first n characters, it returns -1.

Additional information

Converts a single multi-byte character to a wide character in the current locale. The wide character, if the multi-byte character string is converted correctly, is stored into the object pointed to by pwc.

See also

[mbtowc\\_l\(\)](#)

[mbtowc\\_l\(\)](#)

Description

Convert multi-byte character to wide character, per locale (POSIX.1).

Prototype

```
int mbtowc_l(      wchar_t * pwc,  
               const char * s,
```

```
size_t    n,  
          locale_t loc);
```

## Parameters

Parameter	Description
<code>pwc</code>	Pointer to object that receives the wide character.
<code>s</code>	Pointer to multi-byte character string.
<code>n</code>	Maximum number of bytes that will be examined.
<code>loc</code>	Locale used to convert the multi-byte character.

## Return value

If `s` is a null pointer, [mbtowc\\_l\(\)](#) returns a nonzero value if multi-byte character encodings are state-dependent in locale `loc`, and zero otherwise.

If `s` is not null and the object that `s` points to is a wide null character, [mbtowc\\_l\(\)](#) returns 0.

If `s` is not null and the object that `s` points to forms a valid multi-byte character, [mbtowc\\_l\(\)](#) returns the length in bytes of the multi-byte character.

If the object that [mbtowc\\_l\(\)](#) points to does not form a valid multi-byte character within the first `n` characters, it returns -1.

## Additional information

Converts a single multi-byte character to a wide character in the locale `loc`. The wide character, if the multi-byte character string is converted correctly, is stored into the object pointed to by `pwc`.

## Notes

Conforms to POSIX.1-2008.

See also

[mbtowc\(\)](#)

[mbstowcs\(\)](#)

## Description

Convert multi-byte string to wide string.

## Prototype

```
size_t mbstowcs(  
    wchar_t * pwcs,  
    const char * s,  
    size_t n);
```

## Parameters

Parameter	Description
pwcs	Pointer to array that receives the wide character string.
s	Pointer to array that contains the multi-byte string.
n	Maximum number of wide characters to write into pwcs.

## Return value

Returns -1 if an invalid multi-byte character is encountered, otherwise returns the number of array elements modified (if any), not including a terminating null wide character.

## Additional information

Converts a sequence of multi-byte characters, in the current locale, that begins in the initial shift state from the array pointed to by s into a sequence of corresponding wide characters and stores not more than n wide characters into the array pointed to by pwcs.

No multi-byte characters that follow a null character (which is converted into a null wide character) will be examined or converted. Each multi-byte character is converted as if by a call to the [mbtowc\(\)](#) function, except that the conversion state of the [mbtowc\(\)](#) function is not affected.

No more than n elements will be modified in the array pointed to by pwcs. If copying takes place between objects that overlap, the behavior is undefined.

## [mbstowcs\\_l\(\)](#)

## Description

Convert multi-byte string to wide string, per locale (POSIX.1).

## Prototype

```
size_t mbstowcs_l(    wchar_t * pwcs,  
                     const char * s,  
                     size_t    n,  
                     locale_t  loc);
```

## Parameters

Parameter	Description
pwcs	Pointer to array that receives the wide character string.
s	Pointer to array that contains the multi-byte string.
n	Maximum number of wide characters to write into pwcs.
loc	Locale to use for conversion.



## Return value

Returns -1 if an invalid multi-byte character is encountered, otherwise returns the number of array elements modified (if any), not including a terminating null wide character.

## Additional information

Converts a sequence of multi-byte characters, in the locale *loc*, that begins in the initial shift state from the array pointed to by *s* into a sequence of corresponding wide characters and stores not more than *n* wide characters into the array pointed to by *pwcs*.

No multi-byte characters that follow a null character (which is converted into a null wide character) will be examined or converted. Each multi-byte character is converted as if by a call to the [mbtowc\(\)](#) function, except that the conversion state of the [mbtowc\(\)](#) function is not affected.

No more than *n* elements will be modified in the array pointed to by *pwcs*. If copying takes place between objects that overlap, the behavior is undefined.

## Notes

Conforms to POSIX.1-2017.

## [mbsrtowcs\(\)](#)

## Description

Convert multi-byte string to wide character string, restartable.

## Prototype

```
size_t mbsrtowcs(    wchar_t    * dst,
                    const char    ** src,
                    size_t        len,
                    mbstate_t     * ps);
```

## Parameters

Parameter	Description
<i>dst</i>	Pointer to object that receives the converted wide characters.
<i>src</i>	Pointer to pointer to multi-byte character string.
<i>len</i>	Maximum number of wide characters that will be written to <i>dst</i> .
<i>ps</i>	Pointer to multi-byte conversion state.

## Return value

The number of wide characters written to *dst* (not including the eventual terminating null character).

## Additional information

Converts a sequence of multi-byte characters, in the current locale, that begins in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters.

If *dst* is not a null pointer, the converted characters are stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multi-byte character, or (if *dst* is not a null pointer) when *len* wide characters have been stored into the array pointed to by *dst*. Each conversion takes place as if by a call to the [mbrtowc\(\)](#) function.

If *dst* is not a null pointer, the pointer object pointed to by *src* is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multi-byte character converted (if any). If conversion stopped due to reaching a terminating null character and if *dst* is not a null pointer, the resulting state described is the initial conversion state.

See also

[mbsrtowcs\\_l\(\)](#), [mbrtowc\(\)](#)

[mbsrtowcs\\_l\(\)](#)

## Description

Convert multi-byte string to wide character string, restartable, per locale (POSIX.1).

## Prototype

```
size_t mbsrtowcs_l(    wchar_t    * dst,
                      const char    ** src,
                      size_t        len,
                      mbstate_t     * ps,
                      locale_t      loc);
```

## Parameters

Parameter	Description
<i>dst</i>	Pointer to object that receives the converted wide characters.
<i>src</i>	Pointer to pointer to multi-byte character string.
<i>len</i>	Maximum number of wide characters that will be written to <i>dst</i> .
<i>ps</i>	Pointer to multi-byte conversion state.
<i>loc</i>	Locale used for conversion.

## Return value

The number of wide characters written to `dst` (not including the eventual terminating null character).

## Additional information

Converts a sequence of multi-byte characters, in the locale `loc`, that begins in the conversion state described by the object pointed to by `ps`, from the array indirectly pointed to by `src` into a sequence of corresponding wide characters.

If `dst` is not a null pointer, the converted characters are stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null character, which is also stored.

Conversion stops earlier in two cases: when a sequence of bytes is encountered that does not form a valid multi-byte character, or (if `dst` is not a null pointer) when `len` wide characters have been stored into the array pointed to by `dst`. Each conversion takes place as if by a call to the [mbrtowc\(\)](#) function.

If `dst` is not a null pointer, the pointer object pointed to by `src` is assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last multi-byte character converted (if any). If conversion stopped due to reaching a terminating null character and if `dst` is not a null pointer, the resulting state described is the initial conversion state.

## Notes

Conforms to POSIX.1-2008.

See also

[mbsrtowcs\(\)](#), [mbrtowc\(\)](#)

[wctomb\(\)](#)

## Description

Convert wide character to multi-byte character.

## Prototype

```
int wctomb(char * s,  
           wchar_t wc);
```

## Parameters

Parameter	Description
<code>s</code>	Pointer to array that receives the multi-byte character.

`wc` Wide character to convert.

#### Return value

Returns the number of bytes stored in the array object. When `wc` is not a valid wide character, an encoding error occurs: `wctomb()` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1)`; the conversion state is unspecified.

#### Additional information

If `s` is a null pointer, `wctomb()` is equivalent to the call `wcrtomb(buf, 0, ps)` where `buf` is an internal buffer.

If `s` is not a null pointer, `wctomb()` determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by `wc` in the current locale, and stores the multi-byte character representation in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` bytes are stored. If `wc` is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

#### `wctomb_l()`

#### Description

Convert wide character to multi-byte character, per locale (POSIX.1).

#### Prototype

```
int wctomb_l(char * s,
             wchar_t wc,
             locale_t loc);
```

#### Parameters

Parameter	Description
<code>s</code>	Pointer to array that receives the multi-byte character.
<code>wc</code>	Wide character to convert.
<code>loc</code>	Locale used for conversion.

#### Return value

Returns the number of bytes stored in the array object. When `wc` is not a valid wide character, an encoding error occurs: `wctomb_l()` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1)`; the conversion state is unspecified.

#### Additional information

If *s* is a null pointer, `wctomb_l()` is equivalent to the call `wctomb_l(buf, 0, ps, loc)` where *buf* is an internal buffer.

If *s* is not a null pointer, `wctomb_l()` determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by *wc* in the locale *loc*, and stores the multi-byte character representation in the array whose first element is pointed to by *s*. At most `MB_CUR_MAX` bytes are stored. If *wc* is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

## Notes

Conforms to POSIX.1-2008.

## `wcstombs()`

### Description

Convert wide string to multi-byte string.

### Prototype

```
size_t wcstombs(    char    * s,  
                   const wchar_t * pwcs,  
                   size_t    n);
```

### Parameters

Parameter	Description
<i>s</i>	Pointer to array that receives the multi-byte string.
<i>pwcs</i>	Pointer to wide character string to convert.
<i>n</i>	Maximum number of bytes to write into <i>s</i> .

### Return value

If a wide character is encountered that does not correspond to a valid multibyte character in the current locale, returns `(size_t)(-1)`. Otherwise, returns the number of bytes written, not including a terminating null character (if any).

### Additional information

Converts a sequence of wide characters in the current locale from the array pointed to by *pwcs* into a sequence of corresponding multi-byte characters that begins in the initial shift state, and stores these multi-byte characters into the array pointed to by *s*, stopping if a multi-byte character would exceed the limit of *n* total bytes or if a null character is stored. Each wide character is converted as if by a call to `wctomb()`, except that the conversion state of `wctomb()` is not affected.

## `wcstombs_l()`

### Description

Convert wide string to multi-byte string.

### Prototype

```
size_t wcstombs_l(      char      * s,  
                        const wchar_t * pwcs,  
                        size_t      n,  
                        locale_t loc);
```

### Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte string.
pwcs	Pointer to wide character string to convert.
n	Maximum number of bytes to write into s.
loc	Locale used for conversion.

### Return value

If a wide character is encountered that does not correspond to a valid multibyte character in the locale loc, returns `(size_t)(-1)`. Otherwise, returns the number of bytes written, not including a terminating null character (if any).

### Additional information

Converts a sequence of wide characters in the locale loc from the array pointed to by pwcs into a sequence of corresponding multi-byte characters that begins in the initial shift state, and stores these multi-byte characters into the array pointed to by s, stopping if a multi-byte character would exceed the limit of n total bytes or if a null character is stored. Each wide character is converted as if by a call to `wctomb()`, except that the conversion state of `wctomb()` is not affected.

## `<string.h>`

The header file `<string.h>` defines functions that operate on arrays that are interpreted as null-terminated strings.

Various methods are used for determining the lengths of the arrays, but in all cases a `char *` or `void *` argument points to the initial (lowest addressed) character of the array. If an array is accessed beyond the end of an object, the behavior is undefined.

Where an argument declared as `size_t n` specifies the length of an array for a function, n can have the value zero on a call to that function. Unless explicitly stated otherwise in the description of a

particular function, pointer arguments must have valid values on a call with a zero size. On such a call, a function that locates a character finds no occurrence, a function that compares two character sequences returns zero, and a function that copies characters copies zero characters.

### Copying functions

Function	Description
<code>memset()</code>	Set memory to character.
<code>memcpy()</code>	Copy memory.
<code>memccpy()</code>	Copy memory, specify terminator (POSIX.1).
<code>mempcpy()</code>	Copy memory (GNU).
<code>memmove()</code>	Copy memory, tolerate overlaps.
<code>strcpy()</code>	Copy string.
<code>strncpy()</code>	Copy string, limit length.
<code>strlcpy()</code>	Copy string, limit length, always zero terminate (BSD).
<code>stpcpy()</code>	Copy string, return end.
<code>stpncpy()</code>	Copy string, limit length, return end.
<code>strcat()</code>	Concatenate strings.
<code>strncat()</code>	Concatenate strings, limit length.
<code>strlcat()</code>	Concatenate strings, limit length, always zero terminate (BSD).
<code>strdup()</code>	Duplicate string (POSIX.1).
<code>strndup()</code>	Duplicate string, limit length (POSIX.1).

#### *memset()*

##### Description

Set memory to character.

##### Prototype

```
void *memset(void * s,
             int    c,
             size_t n);
```

##### Parameters

Parameter	Description
<code>s</code>	Pointer to destination object.
<code>c</code>	Character to copy.
<code>n</code>	Length of destination object in characters.

Return value

Returns s.

Additional information

Copies the value of c (converted to an unsigned char) into each of the first n characters of the object pointed to by s.

*memcpy()*

Description

Copy memory.

Prototype

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

Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of characters to copy.

Return value

Returns a pointer to the destination object.

Additional information

Copies n characters from the object pointed to by s2 into the object pointed to by s1. The behavior of *memcpy()* is undefined if copying takes place between objects that overlap.

*memccpy()*

Description

Copy memory, specify terminator (POSIX.1).

Prototype

```
void *memccpy(    void    * s1,
                 const void * s2,
```



```
int      c,  
size_t   n);
```

## Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
c	Character that terminates copy.
n	Maximum number of characters to copy.

## Return value

Returns a pointer to the character immediately following c in s1, or NULL if c was not found in the first n characters of s2.

## Additional information

Copies at most n characters from the object pointed to by s2 into the object pointed to by s1. The copying stops as soon as n characters are copied or the character c is copied into the destination object pointed to by s1.

The behavior of `memcpy()` is undefined if copying takes place between objects that overlap.

## Notes

Conforms to POSIX.1-2008.

## `memcpy()`

## Description

Copy memory (GNU).

## Prototype

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

## Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of characters to copy.

## Return value

Returns a pointer to the character immediately following the final character written into s1.

## Additional information

Copies n characters from the object pointed to by s2 into the object pointed to by s1. The behavior of `mempcpy()` is undefined if copying takes place between objects that overlap.

## Notes

This is an extension found in GNU libc.

## `memmove()`

## Description

Copy memory, tolerate overlaps.

## Prototype

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

## Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of characters to copy.

## Return value

Returns the value of s1.

## Additional information

Copies n characters from the object pointed to by s2 into the object pointed to by s1 ensuring that if s1 and s2 overlap, the copy works correctly. Copying takes place as if the n characters from the object pointed to by s2 are first copied into a temporary array of n characters that does not overlap the objects pointed to by s1 and s2, and then the n characters from the temporary array are copied into the object pointed to by s1.

## `strcpy()`

## Description

Copy string.

Prototype

```
char *strcpy(      char * s1,  
                const char * s2);
```

Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.

Return value

Returns the value of s1.

Additional information

Copies the string pointed to by s2 (including the terminating null character) into the array pointed to by s1. The behavior of `strcpy()` is undefined if copying takes place between objects that overlap.

*strncpy()*

Description

Copy string, limit length.

Prototype

```
char *strncpy(      char * s1,  
                  const char * s2,  
                  size_t  n);
```

Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.
n	Maximum number of characters to copy.

Return value

Returns the value of s1.

Additional information

Copies not more than *n* characters from the array pointed to by *s2* to the array pointed to by *s1*. Characters that follow a null character in *s2* are not copied. The behavior of `strncpy()` is undefined if copying takes place between objects that overlap. If the array pointed to by *s2* is a string that is shorter than *n* characters, null characters are appended to the copy in the array pointed to by *s1*, until *n* characters in all have been written.

#### Notes

No null character is implicitly appended to the end of *s1*, so *s1* will only be terminated by a null character if the length of the string pointed to by *s2* is less than *n*.

#### `strncpy()`

#### Description

Copy string, limit length, always zero terminate (BSD).

#### Prototype

```
size_t strncpy(    char    * s1,
                  const char * s2,
                  size_t   n);
```

#### Parameters

Parameter	Description
<i>s1</i>	Pointer to string to copy to.
<i>s2</i>	Pointer to string to copy.
<i>n</i>	Maximum number of characters, including terminating null, in <i>s1</i> .

#### Return value

Returns the number of characters it tried to copy, which is the length of the string *s2* or *n*, whichever is smaller.

#### Additional information

Copies up to *n*-1 characters from the string pointed to by *s2* into the array pointed to by *s1* and always terminates the result with a null character.

The behavior of `strncpy()` is undefined if copying takes place between objects that overlap.

#### Notes

Commonly found in BSD libraries and contrasts with `strncpy()` in that the resulting string is always terminated with a null character.

## *strcpy()*

### Description

Copy string, return end.

### Prototype

```
char *strcpy(      char * s1,  
                const char * s2);
```

### Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.

### Return value

A pointer to the end of the string s1, i.e. the terminating null byte of the string s1, after s2 is copied to it.

### Additional information

Copies the string pointed to by s2 (including the terminating null character) into the array pointed to by s1. The behavior of *strcpy()* is undefined if copying takes place between objects that overlap.

## *strncpy()*

### Description

Copy string, limit length, return end.

### Prototype

```
char *strncpy(      char * s1,  
                  const char * s2,  
                  size_t  n);
```

### Parameters

Parameter	Description
s1	String to copy to.
s2	String to copy.
n	Maximum number of characters to copy.

### Return value

`stpncpy()` returns a pointer to the terminating null byte in `s1` after it is copied to, or, if `s1` is not null-terminated, `s1+n`.

#### Additional information

Copies not more than `n` characters from the array pointed to by `s2` to the array pointed to by `s1`. Characters that follow a null character in `s2` are not copied. The behavior of `stpncpy()` is undefined if copying takes place between objects that overlap. If the array pointed to by `s2` is a string that is shorter than `n` characters, null characters are appended to the copy in the array pointed to by `s1`, until `n` characters in all have been written.

#### Notes

No null character is implicitly appended to the end of `s1`, so `s1` will only be terminated by a null character if the length of the string pointed to by `s2` is less than `n`.

#### `strcat()`

##### Description

Concatenate strings.

##### Prototype

```
char *strcat(      char * s1,  
                  const char * s2);
```

##### Parameters

Parameter	Description
<code>s1</code>	Zero-terminated string to append to.
<code>s2</code>	Zero-terminated string to append.

##### Return value

Returns the value of `s1`.

##### Additional information

Appends a copy of the string pointed to by `s2` (including the terminating null character) to the end of the string pointed to by `s1`. The initial character of `s2` overwrites the null character at the end of `s1`. The behavior of `strcat()` is undefined if copying takes place between objects that overlap.

#### `strncat()`

##### Description

Concatenate strings, limit length.

## Prototype

```
char *strncat(    char    * s1,
                  const char * s2,
                  size_t   n);
```

## Parameters

Parameter	Description
s1	String to append to.
s2	String to append.
n	Maximum number of characters in s1.

## Return value

Returns the value of s1.

## Additional information

Appends not more than n characters from the array pointed to by s2 to the end of the string pointed to by s1. A null character in s1 and characters that follow it are not appended. The initial character of s2 overwrites the null character at the end of s1. A terminating null character is always appended to the result.

The behavior of `strncat()` is undefined if copying takes place between objects that overlap.

## *strlcat()*

## Description

Concatenate strings, limit length, always zero terminate (BSD).

## Prototype

```
size_t strlcat(    char    * s1,
                  const char * s2,
                  size_t   n);
```

## Parameters

Parameter	Description
s1	Pointer to string to append to.
s2	Pointer to string to append.
n	Maximum number of characters, including terminating null, in s1.

## Return value

Returns the number of characters it tried to copy, which is the sum of the lengths of the strings `s1` and `s2` or `n`, whichever is smaller.

#### Additional information

Appends no more than `n-strlen(s1)-1` characters pointed to by `s2` into the array pointed to by `s1` and always terminates the result with a null character if `n` is greater than zero. Both the strings `s1` and `s2` must be terminated with a null character on entry to `strlcat()` and a character position for the terminating null should be included in `n`.

The behavior of `strlcat()` is undefined if copying takes place between objects that overlap.

#### Notes

Commonly found in BSD libraries.

#### `strdup()`

##### Description

Duplicate string (POSIX.1).

##### Prototype

```
char *strdup(const char * s1);
```

##### Parameters

Parameter	Description
<code>s1</code>	Pointer to string to duplicate.

##### Return value

Returns a pointer to the new string or a null pointer if the new string cannot be created. The returned pointer can be passed to `free()`.

#### Additional information

Duplicates the string pointed to by `s1` by using `malloc()` to allocate memory for a copy of `s` and then copies `s`, including the terminating null, to that memory

#### Notes

Conforms to POSIX.1-2008 and SC22 TR 24731-2.

#### `strndup()`

##### Description



Duplicate string, limit length (POSIX.1).

Prototype

```
char *strndup(const char * s,  
              size_t n);
```

Parameters

Parameter	Description
s	Pointer to string to duplicate.
n	Maximum number of characters to duplicate.

Return value

Returns a pointer to the new string or a null pointer if the new string cannot be created. The returned pointer can be passed to [free\(\)](#).

Additional information

Duplicates at most n characters from the the string pointed to by s by using [malloc\(\)](#) to allocate memory for a copy of s.

If the length of string pointed to by s is greater than n characters, only n characters will be duplicated. If n is greater than the length of the string pointed to by s, all characters in the string are copied into the allocated array including the terminating null character.

Notes

Conforms to POSIX.1-2008 and SC22 TR 24731-2.

### Comparison functions

Function	Description
<a href="#">memcmp()</a>	Compare memory.
<a href="#">strcmp()</a>	Compare strings.
<a href="#">strncmp()</a>	Compare strings, limit length.
<a href="#">strcasecmp()</a>	Compare strings, ignore case (POSIX.1).
<a href="#">strncasecmp()</a>	Compare strings, ignore case, limit length (POSIX.1).

[memcmp\(\)](#)

Description

Compare memory.

## Prototype

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

## Parameters

Parameter	Description
s1	Pointer to object #1.
s2	Pointer to object #2.
n	Number of characters to compare.

## Return value

- < 0 s1 is less than s2.
- = 0 s1 is equal to s2.
- > 0 s1 is greater than to s2.

## Additional information

Compares the first n characters of the object pointed to by s1 to the first n characters of the object pointed to by s2. [memcmp\(\)](#) returns an integer greater than, equal to, or less than zero as the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2.

## [strcmp\(\)](#)

## Description

Compare strings.

## Prototype

```
int strcmp(const char * s1,
           const char * s2);
```

## Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.

## Return value

Returns an integer greater than, equal to, or less than zero, if the null-terminated array pointed to by s1 is greater than, equal to, or less than the null-terminated array pointed to by s2.

### *strncmp()*

#### Description

Compare strings, limit length.

#### Prototype

```
int strncmp(const char * s1,
            const char * s2,
            size_t n);
```

#### Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.
n	Maximum number of characters to compare.

#### Return value

Returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

#### Additional information

Compares not more than n characters from the array pointed to by s1 to the array pointed to by s2. Characters that follow a null character are not compared.

### *strcasecmp()*

#### Description

Compare strings, ignore case (POSIX.1).

#### Prototype

```
int strcasecmp(const char * s1,
               const char * s2);
```

#### Parameters

Parameter	Description
-----------	-------------

s1            Pointer to string #1.  
s2            Pointer to string #2.

#### Return value

< 0   s1 is less than s2.  
= 0   s1 is equal to s2.  
> 0   s1 is greater than to s2.

#### Additional information

Compares the string pointed to by s1 to the string pointed to by s2 ignoring differences in case.

[strncasecmp\(\)](#) returns an integer greater than, equal to, or less than zero if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2.

#### Notes

Conforms to POSIX.1-2008.

#### [strncasecmp\(\)](#)

#### Description

Compare strings, ignore case, limit length (POSIX.1).

#### Prototype

```
int strncasecmp(const char * s1,  
               const char * s2,  
               size_t  n);
```

#### Parameters

Parameter	Description
s1	Pointer to string #1.
s2	Pointer to string #2.
n	Maximum number of characters to compare.

#### Return value

< 0   s1 is less than s2.  
= 0   s1 is equal to s2.  
> 0   s1 is greater than to s2.

## Additional information

Compares not more than *n* characters from the array pointed to by *s1* to the array pointed to by *s2* ignoring differences in case. Characters that follow a null character are not compared.

[`strncasecmp\(\)`](#) returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*.

## Notes

Conforms to POSIX.1-2008.

## Search functions

Function	Description
<a href="#"><code>memchr()</code></a>	Find character in memory, forward.
<a href="#"><code>memrchr()</code></a>	Find character in memory, reverse (BSD).
<a href="#"><code>memmem()</code></a>	Find memory in memory, forward (BSD).
<a href="#"><code>strchr()</code></a>	Find character within string, forward.
<a href="#"><code>strnchr()</code></a>	Find character within string, forward, limit length.
<a href="#"><code>strrchr()</code></a>	Find character within string, reverse.
<a href="#"><code>strlen()</code></a>	Calculate length of string.
<a href="#"><code>strnlen()</code></a>	Calculate length of string, limit length (POSIX.1).
<a href="#"><code>strstr()</code></a>	Find string within string, forward.
<a href="#"><code>strnstr()</code></a>	Find string within string, forward, limit length (BSD).
<a href="#"><code>strcasestr()</code></a>	Find string within string, forward, ignore case (BSD).
<a href="#"><code>strncasestr()</code></a>	Find string within string, forward, ignore case, limit length (BSD).
<a href="#"><code>strpbrk()</code></a>	Find first occurrence of characters within string.
<a href="#"><code>strspn()</code></a>	Compute size of string prefixed by a set of characters.
<a href="#"><code>strcspn()</code></a>	Compute size of string not prefixed by a set of characters.
<a href="#"><code>strtok()</code></a>	Break string into tokens.
<a href="#"><code>strtok_r()</code></a>	Break string into tokens, reentrant (POSIX.1).
<a href="#"><code>strsep()</code></a>	Break string into tokens (BSD).

## [`memchr\(\)`](#)

## Description

Find character in memory, forward.

## Prototype

```
void *memchr(const void * s,  
             int c,  
             size_t n);
```

## Parameters

Parameter	Description
s	Pointer to object to search.
c	Character to search for.
n	Number of characters in object to search.

## Return value

- = NULL c does not occur in the object.
- ≠ NULL Pointer to the located character.

## Additional information

Locates the first occurrence of c (converted to an unsigned char) in the initial n characters (each interpreted as unsigned char) of the object pointed to by s. Unlike [strchr\(\)](#), [memchr\(\)](#) does not terminate a search when a null character is found in the object pointed to by s.

## [memrchr\(\)](#)

## Description

Find character in memory, reverse (BSD).

## Prototype

```
void *memrchr(const void * s,  
             int c,  
             size_t n);
```

## Parameters

Parameter	Description
s	Pointer to object to search.
c	Character to search for.
n	Number of characters in object to search.

## Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

## Additional information

Locates the last occurrence of *c* (converted to a char) in the string pointed to by *s*.

## Notes

Commonly found in Linux and BSD C libraries.

## *memmem()*

## Description

Find memory in memory, forward (BSD).

## Prototype

```
void *memmem(const void * s1,
             size_t   n1,
             const void * s2,
             size_t   n2);
```

## Parameters

Parameter	Description
<i>s1</i>	Pointer to object to search.
<i>n1</i>	Number of characters to search in <i>s1</i> .
<i>s2</i>	Pointer to object to search for.
<i>n2</i>	Number of characters to search from <i>s2</i> .

## Return value

= NULL (s2, n2) does not occur in (s1, n1).

≠ NULL Pointer to the first occurrence of (s2, n2) in (s1, n1).

## Additional information

Locates the first occurrence of the octet string *s2* of length *n2* in the octet string *s1* of length *n1*.

## Notes

Commonly found in Linux and BSD C libraries.

## *strchr()*

## Description

Find character within string, forward.

## Prototype

```
char *strchr(const char * s,  
             int      c);
```

## Parameters

Parameter	Description
s	String to search.
c	Character to search for.

## Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

## Additional information

Locates the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.

## *strnchr()*

## Description

Find character within string, forward, limit length.

## Prototype

```
char *strnchr(const char * s,  
              size_t    n,  
              int      c);
```

## Parameters

Parameter	Description
s	String to search.
n	Number of characters to search.
c	Character to search for.

## Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

## Additional information

Searches not more than n characters to locate the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.



## *strrchr()*

### Description

Find character within string, reverse.

### Prototype

```
char *strrchr(const char * s,  
              int      c);
```

### Parameters

Parameter	Description
s	String to search.
c	Character to search for.

### Return value

Returns a pointer to the located character, or a null pointer if c does not occur in the string.

### Additional information

Locates the last occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string.

## *strlen()*

### Description

Calculate length of string.

### Prototype

```
size_t strlen(const char * s);
```

### Parameters

Parameter	Description
s	Pointer to zero-terminated string.

### Return value

Returns the length of the string pointed to by s, that is the number of characters that precede the terminating null character.

## *strlen()*

### Description

Calculate length of string, limit length (POSIX.1).

### Prototype

```
size_t strlen(const char * s,  
              size_t n);
```

### Parameters

Parameter	Description
s	Pointer to string.
n	Maximum number of characters to examine.

### Return value

Returns the length of the string pointed to by s, up to a maximum of n characters. *strlen()* only examines the first n characters of the string s.

### Notes

Conforms to POSIX.1-2008.

## *strstr()*

### Description

Find string within string, forward.

### Prototype

```
char *strstr(const char * s1,  
            const char * s2);
```

### Parameters

Parameter	Description
s1	String to search.
s2	String to search for.

### Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, *strstr()* returns s1.

## Additional information

Locates the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2.

### *strnstr()*

## Description

Find string within string, forward, limit length (BSD).

## Prototype

```
char *strnstr(const char * s1,  
              const char * s2,  
              size_t n);
```

## Parameters

Parameter	Description
s1	String to search.
s2	String to search for.
n	Maximum number of characters to search for.

## Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, *strnstr()* returns s1.

## Additional information

Searches at most n characters to locate the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2.

## Notes

Commonly found in Linux and BSD C libraries.

### *strcasestr()*

## Description

Find string within string, forward, ignore case (BSD).

## Prototype

```
char *strcasestr(const char * s1,  
                 const char * s2);
```

## Parameters

Parameter	Description
s1	String to search for.
s2	String to search.

## Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, returns s1.

## Additional information

Locates the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2 without regard to character case.

## Notes

This extension is commonly found in Linux and BSD C libraries.

## *strncasestr()*

## Description

Find string within string, forward, ignore case, limit length (BSD).

## Prototype

```
char *strncasestr(const char * s1,  
                  const char * s2,  
                  size_t n);
```

## Parameters

Parameter	Description
s1	String to search for.
s2	String to search.
n	Maximum number of characters to compare in s2.

## Return value

Returns a pointer to the located string, or a null pointer if the string is not found. If s2 points to a string with zero length, returns s1.

## Additional information

Searches at most n characters to locate the first occurrence in the string pointed to by s1 of the sequence of characters (excluding the terminating null character) in the string pointed to by s2 without regard to character case.

#### Notes

This extension is commonly found in Linux and BSD C libraries.

#### *strpbrk()*

##### Description

Find first occurrence of characters within string.

##### Prototype

```
char *strpbrk(const char * s1,  
              const char * s2);
```

##### Parameters

Parameter	Description
s1	Pointer to string to search.
s2	Pointer to string to search for.

##### Return value

Returns a pointer to the first character, or a null pointer if no character from s2 occurs in s1.

##### Additional information

Locates the first occurrence in the string pointed to by s1 of any character from the string pointed to by s2.

#### *strspn()*

##### Description

Compute size of string prefixed by a set of characters.

##### Prototype

```
size_t strspn(const char * s1,  
              const char * s2);
```

##### Parameters

Parameter	Description
-----------	-------------

s1            Pointer to zero-terminated string to search.  
s2            Pointer to zero-terminated acceptable-set string.

#### Return value

Returns the length of the string pointed to by s1 which consists entirely of characters from the string pointed to by s2

#### Additional information

Computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters from the string pointed to by s2.

### *strcspn()*

#### Description

Compute size of string not prefixed by a set of characters.

#### Prototype

```
size_t strcspn(const char * s1,  
               const char * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to string to search.
s2	Pointer to string containing characters to skip.

#### Return value

Returns the length of the segment of string s1 prefixed by characters from s2.

#### Additional information

Computes the length of the maximum initial segment of the string pointed to by s1 which consists entirely of characters not from the string pointed to by s2.

### *strtok()*

#### Description

Break string into tokens.

#### Prototype

```
char *strtok(      char * s1,  
                const char * s2);
```

## Parameters

Parameter	Description
s1	Pointer to zero-terminated string to parse.
s2	Pointer to zero-terminated set of separators.

## Return value

NULL if no further tokens else a pointer to the next token.

## Additional information

A sequence of calls to [strtok\(\)](#) breaks the string pointed to by s1 into a sequence of tokens, each of which is delimited by a character from the string pointed to by s2. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator string pointed to by s2 may be different from call to call.

The first call in the sequence searches the string pointed to by s1 for the first character that is not contained in the current separator string pointed to by s2. If no such character is found, then there are no tokens in the string pointed to by s1 and [strtok\(\)](#) returns a null pointer. If such a character is found, it is the start of the first token.

[strtok\(\)](#) then searches from there for a character that is contained in the current separator string. If no such character is found, the current token extends to the end of the string pointed to by s1, and subsequent searches for a token will return a null pointer. If such a character is found, it is overwritten by a null character, which terminates the current token. [strtok\(\)](#) saves a pointer to the following character, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

## Notes

[strtok\(\)](#) maintains static state and is therefore not reentrant and not thread safe. See [strtok\\_r\(\)](#) for a thread-safe and reentrant variant.

See also

[strsep\(\)](#), [strtok\\_r\(\)](#).

[strtok\\_r\(\)](#)

## Description

Break string into tokens, reentrant (POSIX.1).

Prototype

```
char *strtok_r(    char * s1,  
                  const char * s2,  
                  char ** lasts);
```

Parameters

Parameter	Description
s1	Pointer to zero-terminated string to parse.
s2	Pointer to zero-terminated set of separators.
lasts	Pointer to pointer to char that maintains parse state.

Return value

NULL if no further tokens else a pointer to the next token.

Additional information

[strtok\\_r\(\)](#) is a reentrant version of the function [strtok\(\)](#) where the state is maintained in the object of type `char *` pointed to by s3.

Notes

Conforms to POSIX.1-2008 and is commonly found in Linux and BSD C libraries.

See also

[strtok\(\)](#)

[strsep\(\)](#)

Description

Break string into tokens (BSD).

Prototype

```
char *strsep(    char ** stringp,  
                const char * delim);
```

Parameters

Parameter	Description
stringp	Pointer to pointer to zero-terminated string.



delim      Pointer to delimiter set string.

Return value

See below.

Additional information

Locates, in the string referenced by *\*stringp*, the first occurrence of any character in the string *delim* (or the terminating null character) and replaces it with a null character. The location of the next character after the delimiter character (or NULL, if the end of the string was reached) is stored in *\*stringp*. The original value of *\*stringp* is returned.

An empty field (that is, a character in the string *delim* occurs as the first character of *\*stringp*) can be detected by comparing the location referenced by the returned pointer to the null wide character.

If *\*stringp* is initially null, [strsep\(\)](#) returns null.

Notes

Commonly found in Linux and BSD C libraries.

### Miscellaneous functions

Function	Description
<a href="#">strerror()</a>	Decode error code.

[strerror\(\)](#)

Description

Decode error code.

Prototype

```
char *strerror(int num);
```

Parameters

Parameter	Description
num	Error number.

Return value

Returns a pointer to the message string. The program must not modify the returned message string. The message may be overwritten by a subsequent call to [strerror\(\)](#).

## Additional information

Maps the number in num to a message string. Typically, the values for num come from errno, but `strerror()` can map any value of type int to a message.

## <time.h>

### Operations

Function	Description
<code>mktime()</code>	Convert a struct tm to time_t.
<code>difftime()</code>	Calculate difference between two times.

### *mktime()*

#### Description

Convert a struct tm to time\_t.

#### Prototype

```
time_t mktime(tm * tp);
```

#### Parameters

Parameter	Description
tp	Pointer to time object.

#### Return value

Number of seconds since UTC 1 January 1970 of the validated object.

#### Additional information

Validates (and updates) the object pointed to by tp to ensure that the tm\_sec, tm\_min, tm\_hour, and tm\_mon fields are within the supported integer ranges and the tm\_mday, tm\_mon and tm\_year fields are consistent. The validated object is converted to the number of seconds since UTC 1 January 1970 and returned.

### *difftime()*

#### Description

Calculate difference between two times.

#### Prototype

```
double difftime(time_t time2,  
                time_t time1);
```

#### Parameters

Parameter	Description
time2	End time.
time1	Start time.

#### Return value

returns time2-time1 as a double precision number.

### Conversion functions

Function	Description
<a href="#">ctime()</a>	Convert time_t to a string.
<a href="#">ctime_r()</a>	Convert time_t to a string, reentrant.
<a href="#">asctime()</a>	Convert time_t to a string.
<a href="#">asctime_r()</a>	Convert time_t to a string, reentrant.
<a href="#">gmtime()</a>	Convert time_t to struct tm.
<a href="#">gmtime_r()</a>	Convert time_t to struct tm, reentrant.
<a href="#">localtime()</a>	Convert time to local time.
<a href="#">localtime_r()</a>	Convert time to local time, reentrant.
<a href="#">strftime()</a>	Convert time to a string.
<a href="#">strftime_l()</a>	Convert time to a string.

#### [ctime\(\)](#)

#### Description

Convert time\_t to a string.

#### Prototype

```
char *ctime(const time_t * tp);
```

#### Parameters

Parameter	Description
tp	Pointer to time to convert.

#### Return value

Pointer to zero-terminated converted string.

Additional information

Converts the time pointed to by tp to a null-terminated string.

Notes

The returned string is held in a static buffer: this function is not thread safe.

*ctime\_r()*

Description

Convert time\_t to a string, reentrant.

Prototype

```
char *ctime_r(const time_t * tp,  
              char * buf);
```

Parameters

Parameter	Description
tp	Pointer to time to convert.
buf	Pointer to array of characters that receives the zero-terminated string; the array must be at least 26 characters in length.

Return value

Returns the value of buf.

Additional information

Converts the time pointed to by tp to a null-terminated string.

Notes

The returned string is held in a static buffer: this function is not thread safe.

*asctime()*

Description

Convert time\_t to a string.

Prototype

```
char *asctime(const tm * tp);
```

## Parameters

Parameter	Description
tp	Pointer to time to convert.

## Return value

Pointer to zero-terminated converted string.

## Additional information

Converts the time pointed to by tp to a null-terminated string of the Sun Sep 16 01:03:52 1973. The returned string is held in a static buffer.

## Notes

The returned string is held in a static buffer: this function is not thread safe.

## [asctime\\_r\(\)](#)

## Description

Convert time\_t to a string, reentrant.

## Prototype

```
char *asctime_r(const tm * tp,  
                char * buf);
```

## Parameters

Parameter	Description
tp	Pointer to time to convert.
buf	Pointer to array of characters that receives the zero-terminated string; the array must be at least 26 characters in length.

## Return value

Returns the value of buf.

## Additional information

Converts the time pointed to by tp to a null-terminated string of the Sun Sep 16 01:03:52 1973. The converted string is written into the array pointed to by buf.

## [gmtime\(\)](#)

## Description

Convert `time_t` to struct `tm`.

Prototype

```
gmtime(const time_t * tp);
```

Parameters

Parameter	Description
<code>tp</code>	Pointer to time to convert.

Return value

Pointer to converted time.

Additional information

Converts the time pointed to by `tp` to a struct `tm`.

Notes

The returned pointer points to a static buffer: this function is not thread safe.

[\*`gmtime\_r\(\)`\*](#)

Description

Convert `time_t` to struct `tm`, reentrant.

Prototype

```
gmtime_r(const time_t * tp,  
         tm *tm);
```

Parameters

Parameter	Description
<code>tp</code>	Pointer to time to convert.
<code>tm</code>	Pointer to object that receives the converted time.

Return value

Returns `tm`.

Additional information

Converts the time pointed to by `tp` to a struct `tm`.

## *localtime()*

### Description

Convert time to local time.

### Prototype

```
localtime(const time_t * tp);
```

### Parameters

Parameter	Description
tp	Pointer to time to convert.

### Return value

Pointer to a statically-allocated object holding the local time.

### Additional information

Converts the time pointed to by tp to local time format.

### Notes

The returned pointer points to a static object: this function is not thread safe.

## *localtime\_r()*

### Description

Convert time to local time, reentrant.

### Prototype

```
localtime_r(const time_t * tp,  
            tm *tm);
```

### Parameters

Parameter	Description
tp	Pointer to time to convert.
tm	Pointer to object that receives the converted local time.

### Return value

Returns tm.

### Additional information

Converts the time pointed to by `tp` to local time format and writes it to the object pointed to by `tm`.

### *strftime()*

#### Description

Convert time to a string.

#### Prototype

```
size_t strftime(    char    * s,  
                   size_t  smax,  
                   const char * fmt,  
                   const tm  * tp);
```

#### Parameters

Parameter	Description
<code>s</code>	Pointer to object that receives the converted string.
<code>smax</code>	Maximum number of characters written to the array pointed to by <code>s</code> .
<code>fmt</code>	Pointer to zero-terminated format control string.
<code>tp</code>	Pointer to time to convert.

#### Return value

Returns the name of the current locale.

#### Additional information

Formats the time pointed to by `tp` to a null-terminated string of maximum size `smax-1` into the pointed to by `*s` based on the `fmt` format string. The format string consists of conversion specifications and ordinary characters. Conversion specifications start with a “%” character followed by an optional “#” character.

The following conversion specifications are supported:

Specification	Description
<code>%a</code>	Abbreviated weekday name
<code>%A</code>	Full weekday name
<code>%b</code>	Abbreviated month name
<code>%B</code>	Full month name
<code>%c</code>	Date and time representation appropriate for locale
<code> %#c</code>	Date and time formatted as “%A, %B %d, %Y, %H:%M:%S” (Microsoft extension)
<code>%C</code>	Century number



---

%d	Day of month as a decimal number [01,31]
%#d	Day of month without leading zero [1,31]
%D	Date in the form %m/%d/%y (POSIX.1-2008 extension)
%e	Day of month [ 1,31], single digit preceded by space
%F	Date in the format %Y-%m-%d
%h	Abbreviated month name as %b
%H	Hour in 24-hour format [00,23]
%#H	Hour in 24-hour format without leading zeros [0,23]
%I	Hour in 12-hour format [01,12]
%#I	Hour in 12-hour format without leading zeros [1,12]
%j	Day of year as a decimal number [001,366]
%#j	Day of year as a decimal number without leading zeros [1,366]
%k	Hour in 24-hour clock format [ 0,23] (POSIX.1-2008 extension)
%l	Hour in 12-hour clock format [ 0,12] (POSIX.1-2008 extension)
%m	Month as a decimal number [01,12]
%#m	Month as a decimal number without leading zeros [1,12]
%M	Minute as a decimal number [00,59]
%#M	Minute as a decimal number without leading zeros [0,59]
%n	Insert newline character (POSIX.1-2008 extension)
%p	Locale's a.m or p.m indicator for 12-hour clock
%r	Time as %I:%M:%s %p (POSIX.1-2008 extension)
%R	Time as %H:%M (POSIX.1-2008 extension)
%S	Second as a decimal number [00,59]
%t	Insert tab character (POSIX.1-2008 extension)
%T	Time as %H:%M:%S
%#S	Second as a decimal number without leading zeros [0,59]
%U	Week of year as a decimal number [00,53], Sunday is first day of the week
%#U	Week of year as a decimal number without leading zeros [0,53], Sunday is first day of the week
%w	Weekday as a decimal number [0,6], Sunday is 0
%W	Week number as a decimal number [00,53], Monday is first day of the week
%#W	Week number as a decimal number without leading zeros [0,53], Monday is first day of the week
%x	Locale's date representation

---

%#x	Locale's long date representation
%X	Locale's time representation
%y	Year without century, as a decimal number [00,99]
%#y	Year without century, as a decimal number without leading zeros [0,99]
%Y	Year with century, as decimal number
%z,%Z	Timezone name or abbreviation
%%	%

## [strftime\\_l\(\)](#)

### Description

Convert time to a string.

### Prototype

```
size_t strftime_l(    char    * s,  
                     size_t  smax,  
                     const char * fmt,  
                     const tm  * tp,  
                     locale_t loc);
```

### Parameters

Parameter	Description
s	Pointer to object that receives the converted string.
smax	Maximum number of characters written to the array pointed to by s.
fmt	Pointer to zero-terminated format control string.
tp	Pointer to time to convert.
loc	Locale to use for conversion.

### Return value

Returns the name of the current locale.

### Additional information

Formats the time pointed to by tp to a null-terminated string of maximum size smax-1 into the pointed to by \*s based on the fmt format string and using the locale loc.

The format string consists of conversion specifications and ordinary characters. Conversion specifications start with a “%” character followed by an optional “#” character.

See [strftime\(\)](#) for a description of the format conversion specifications.

## <wchar.h>

### Copying functions

Function	Description
<a href="#">wmemset()</a>	Set memory to wide character.
<a href="#">wmemcpy()</a>	Copy memory.
<a href="#">wmemccpy()</a>	Copy memory, specify terminator (POSIX.1).
<a href="#">wmempcpy()</a>	Copy memory (GNU).
<a href="#">wmemmove()</a>	Copy memory, tolerate overlaps.
<a href="#">wcscpy()</a>	Copy string.
<a href="#">wcsncpy()</a>	Copy string, limit length.
<a href="#">wcsncpy()</a>	Copy string, limit length, always zero terminate (BSD).
<a href="#">wcscat()</a>	Concatenate strings.
<a href="#">wcsncat()</a>	Concatenate strings, limit length.
<a href="#">wcsncat()</a>	Concatenate strings, limit length, always zero terminate (BSD).
<a href="#">wcsdup()</a>	Duplicate string (POSIX.1).
<a href="#">wcsndup()</a>	Duplicate string, limit length (GNU).

### *wmemset()*

#### Description

Set memory to wide character.

#### Prototype

```
wchar_t *wmemset(wchar_t * s,  
                 wchar_t  c,  
                 size_t   n);
```

#### Parameters

Parameter	Description
s	Pointer to destination object.
c	Wide character to copy.
n	Length of destination object in wide characters.

#### Return value

Returns s.

## Additional information

Copies the value of *c* into each of the first *n* wide characters of the object pointed to by *s*.

### *wmemcpy()*

## Description

Copy memory.

## Prototype

```
wchar_t *wmemcpy(    wchar_t * s1,  
                    const wchar_t * s2,  
                    size_t    n);
```

## Parameters

Parameter	Description
<i>s1</i>	Pointer to destination object.
<i>s2</i>	Pointer to source object.
<i>n</i>	Number of wide characters to copy.

## Return value

Returns the value of *s1*.

## Additional information

Copies *n* wide characters from the object pointed to by *s2* into the object pointed to by *s1*. The behavior of *wmemcpy()* is undefined if copying takes place between objects that overlap.

### *wmemccpy()*

## Description

Copy memory, specify terminator (POSIX.1).

## Prototype

```
wchar_t *wmemccpy(    wchar_t * s1,  
                    const wchar_t * s2,  
                    wchar_t    c,  
                    size_t    n);
```

## Parameters

Parameter	Description
-----------	-------------

s1	Pointer to destination object.
s2	Pointer to source object.
c	Character that terminates copy.
n	Maximum number of characters to copy.

#### Return value

Returns a pointer to the wide character immediately following c in s1, or NULL if c was not found in the first n wide characters of s2.

#### Additional information

Copies at most n wide characters from the object pointed to by s2 into the object pointed to by s1. The copying stops as soon as n wide characters are copied or the wide character c is copied into the destination object pointed to by s1.

The behavior of `wmemccpy()` is undefined if copying takes place between objects that overlap.

#### Notes

Conforms to POSIX.1-2008.

#### `wmempcpy()`

#### Description

Copy memory (GNU).

#### Prototype

```
wchar_t *wmempcpy(    wchar_t * s1,  
                     const wchar_t * s2,  
                     size_t      n);
```

#### Parameters

Parameter	Description
s1	Pointer to destination object.
s2	Pointer to source object.
n	Number of wide characters to copy.

#### Return value

Returns a pointer to the wide character immediately following the final wide character written into s1.

## Additional information

Copies *n* wide characters from the object pointed to by *s2* into the object pointed to by *s1*. The behavior of `wmempcpy()` is undefined if copying takes place between objects that overlap.

## Notes

This is an extension found in GNU libc.

## `wmemmove()`

## Description

Copy memory, tolerate overlaps.

## Prototype

```
wchar_t *wmemmove(    wchar_t * s1,
                      const wchar_t * s2,
                      size_t      n);
```

## Parameters

Parameter	Description
<i>s1</i>	Pointer to destination object.
<i>s2</i>	Pointer to source object.
<i>n</i>	Number of wide characters to copy.

## Return value

Returns the value of *s1*.

## Additional information

Copies *n* wide characters from the object pointed to by *s2* into the object pointed to by *s1* ensuring that if *s1* and *s2* overlap, the copy works correctly. Copying takes place as if the *n* wide characters from the object pointed to by *s2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *s1* and *s2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *s1*.

## `wcscpy()`

## Description

Copy string.

## Prototype

```
wchar_t *wcscpy(    wchar_t * s1,  
                  const wchar_t * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string to copy to.
s2	Pointer to wide string to copy.

#### Return value

Returns the value of s1.

#### Additional information

Copies the wide string pointed to by s2 (including the terminating null wide character) into the array pointed to by s1. The behavior of [wcscpy\(\)](#) is undefined if copying takes place between objects that overlap.

#### [wcsncpy\(\)](#)

#### Description

Copy string, limit length.

#### Prototype

```
wchar_t *wcsncpy(    wchar_t * s1,  
                  const wchar_t * s2,  
                  size_t    n);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string to copy to.
s2	Pointer to wide string to copy.
n	Maximum number of wide characters to copy.

#### Return value

Returns the value of s1.

#### Additional information

Copies not more than n wide characters from the array pointed to by s2 to the array pointed to by s1. Wide characters that follow a null wide character in s2 are not copied. The behavior of [wcsncpy\(\)](#) is undefined if copying takes place between objects that overlap. If the array pointed to

by s2 is a wide string that is shorter than n wide characters, null wide characters are appended to the copy in the array pointed to by s1, until n characters in all have been written.

#### Notes

No wide null character is implicitly appended to the end of s1, so s1 will only be terminated by a wide null character if the length of the wide string pointed to by s2 is less than n.

#### [wcsncpy\(\)](#)

##### Description

Copy string, limit length, always zero terminate (BSD).

##### Prototype

```
size_t wcsncpy(    wchar_t * s1,
                  const wchar_t * s2,
                  size_t  n);
```

##### Parameters

Parameter	Description
s1	Pointer to wide string to copy to.
s2	Pointer to wide string to copy.
n	Maximum number of wide characters, including terminating null, in s1.

##### Return value

Returns the number of wide characters it tried to copy, which is the length of the wide string s2 or n, whichever is smaller.

##### Additional information

Copies up to n-1 wide characters from the wide string pointed to by s2 into the array pointed to by s1 and always terminates the result with a null character.

The behavior of [strcpy\(\)](#) is undefined if copying takes place between objects that overlap.

#### Notes

Commonly found in BSD libraries and contrasts with [wcsncpy\(\)](#) in that the resulting string is always terminated with a null wide character.

#### [wcscat\(\)](#)

##### Description



Concatenate strings.

Prototype

```
wchar_t *wcscat(    wchar_t * s1,  
                   const wchar_t * s2);
```

Parameters

Parameter	Description
s1	Zero-terminated wide string to append to.
s2	Zero-terminated wide string to append.

Return value

Returns the value of s1.

Additional information

Appends a copy of the wide string pointed to by s2 (including the terminating null wide character) to the end of the wide string pointed to by s1. The initial character of s2 overwrites the null wide character at the end of s1. The behavior of `wcscat()` is undefined if copying takes place between objects that overlap.

[`wcsncat\(\)`](#)

Description

Concatenate strings, limit length.

Prototype

```
wchar_t *wcsncat(    wchar_t * s1,  
                   const wchar_t * s2,  
                   size_t    n);
```

Parameters

Parameter	Description
s1	Wide string to append to.
s2	Wide string to append.
n	Maximum number of wide characters in s1.

Return value

Returns the value of s1.

## Additional information

Appends not more than *n* wide characters from the array pointed to by *s2* to the end of the wide string pointed to by *s1*. A null wide character in *s1* and wide characters that follow it are not appended. The initial wide character of *s2* overwrites the null wide character at the end of *s1*. A terminating wide null character is always appended to the result. The behavior of `wcsncat()` is undefined if copying takes place between objects that overlap.

## `wcslcat()`

### Description

Concatenate strings, limit length, always zero terminate (BSD).

### Prototype

```
size_t wclcat(      char    * s1,  
                  const char * s2,  
                  size_t   n);
```

### Parameters

Parameter	Description
<i>s1</i>	Pointer to wide string to append to.
<i>s2</i>	Pointer to wide string to append.
<i>n</i>	Maximum number of characters, including terminating wide null, in <i>s1</i> .

### Return value

Returns the number of wide characters it tried to copy, which is the sum of the lengths of the wide strings *s1* and *s2* or *n*, whichever is smaller.

## Additional information

Appends no more than `n-strlen(s1)-1` wide characters pointed to by *s2* into the array pointed to by *s1* and always terminates the result with a wide null character if *n* is greater than zero. Both the wide strings *s1* and *s2* must be terminated with a wide null character on entry to `wclcat()` and a character position for the terminating wide null should be included in *n*.

The behavior of `wclcat()` is undefined if copying takes place between objects that overlap.

### Notes

Commonly found in BSD libraries.

## *wcsdup()*

### Description

Duplicate string (POSIX.1).

### Prototype

```
wchar_t *wcsdup(const wchar_t * s);
```

### Parameters

Parameter	Description
s	Pointer to wide string to duplicate.

### Return value

Returns a pointer to the new wide string or a null pointer if the new wide string cannot be created. The returned pointer can be passed to [free\(\)](#).

### Additional information

Duplicates the wide string pointed to by s by using [malloc\(\)](#) to allocate memory for a copy of s and then copies s, including the terminating null, to that memory

### Notes

Conforms to POSIX.1-2008 and SC22 TR 24731-2.

## *wcsndup()*

### Description

Duplicate string, limit length (GNU).

### Prototype

```
wchar_t *wcsndup(const wchar_t * s,  
                 size_t      n);
```

### Parameters

Parameter	Description
s	Pointer to wide string to duplicate.
n	Maximum number of wide characters to duplicate.

### Return value

Returns a pointer to the new wide string or a null pointer if the new wide string cannot be created. The returned pointer can be passed to [free\(\)](#).

#### Additional information

Duplicates at most *n* wide characters from the the string pointed to by *s* by using [malloc\(\)](#) to allocate memory for a copy of *s*.

If the length of string pointed to by *s* is greater than *n* wide characters, only *n* wide characters will be duplicated. If *n* is greater than the length of the wide string pointed to by *s*, all characters in the string are copied into the allocated array including the terminating null character.

#### Notes

This is a GNU extension.

### Comparison functions

Function	Description
<a href="#">wmemcmp()</a>	Compare memory.
<a href="#">wcsncmp()</a>	Compare strings, limit length.
<a href="#">wcscasecmp()</a>	Compare strings, ignore case (POSIX.1).
<a href="#">wcsncasecmp()</a>	Compare strings, ignore case, limit length (POSIX.1).

#### [wmemcmp\(\)](#)

#### Description

Compare memory.

#### Prototype

```
int wmemcmp(const wchar_t * s1,
            const wchar_t * s2,
            size_t      n);
```

#### Parameters

Parameter	Description
<i>s1</i>	Pointer to object #1.
<i>s2</i>	Pointer to object #2.
<i>n</i>	Number of wide characters to compare.

#### Return value

< 0   *s1* is less than *s2*.

= 0 s1 is equal to s2.  
> 0 s1 is greater than to s2.

#### Additional information

Compares the first n wide characters of the object pointed to by s1 to the first n wide characters of the object pointed to by s2. `wmemcmp()` returns an integer greater than, equal to, or less than zero as the object pointed to by s1 is greater than, equal to, or less than the object pointed to by s2.

#### `wcsncmp()`

##### Description

Compare strings, limit length.

##### Prototype

```
int wcsncmp(const wchar_t * s1,
            const wchar_t * s2,
            size_t      n);
```

##### Parameters

Parameter	Description
s1	Pointer to wide string #1.
s2	Pointer to wide string #2.
n	Maximum number of wide characters to compare.

##### Return value

Returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

#### Additional information

Compares not more than n wide characters from the array pointed to by s1 to the array pointed to by s2. Wide characters that follow a null wide character are not compared.

#### `wscasecmp()`

##### Description

Compare strings, ignore case (POSIX.1).

##### Prototype

```
int wcscasecmp(const char * s1,
               const char * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string #1.
s2	Pointer to wide string #2.

#### Return value

< 0 s1 is less than s2.  
= 0 s1 is equal to s2.  
> 0 s1 is greater than to s2.

#### Additional information

Compares the wide string pointed to by s1 to the wide string pointed to by s2 ignoring differences in case.

[wcscasecmp\(\)](#) returns an integer greater than, equal to, or less than zero if the wide string pointed to by s1 is greater than, equal to, or less than the wide string pointed to by s2.

#### Notes

Conforms to POSIX.1-2017.

#### [wcsncasecmp\(\)](#)

#### Description

Compare strings, ignore case, limit length (POSIX.1).

#### Prototype

```
int wcsncasecmp(const char * s1,
               const char * s2,
               size_t n);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string #1.
s2	Pointer to wide string #2.
n	Maximum number of wide characters to compare.

## Return value

- < 0 s1 is less than s2.
- = 0 s1 is equal to s2.
- > 0 s1 is greater than to s2.

## Additional information

Compares not more than n wide characters from the array pointed to by s1 to the array pointed to by s2 ignoring differences in case. Characters that follow a wide null character are not compared.

[strncasecmp\(\)](#) returns an integer greater than, equal to, or less than zero, if the possibly null-terminated array pointed to by s1 is greater than, equal to, or less than the possibly null-terminated array pointed to by s2.

## Notes

Conforms to POSIX.1-2017.

## Search functions

Function	Description
<a href="#">wmemchr()</a>	Find character in memory, forward.
<a href="#">wcschr()</a>	Find character within string, forward.
<a href="#">wcsnchr()</a>	Find character within string, forward, limit length.
<a href="#">wcsrchr()</a>	Find character within string, reverse.
<a href="#">wcslen()</a>	Calculate length of string.
<a href="#">wcsnlen()</a>	Calculate length of string, limit length (POSIX.1).
<a href="#">wcsstr()</a>	Find string within string, forward.
<a href="#">wcsnstr()</a>	Find string within string, forward, limit length (BSD).
<a href="#">wcsbrk()</a>	Find first occurrence of characters within string.
<a href="#">wcssp()</a>	Compute size of string prefixed by a set of characters.
<a href="#">wcscsp()</a>	Compute size of string not prefixed by a set of characters.
<a href="#">wcstok()</a>	Break string into tokens.
<a href="#">wcssep()</a>	Break string into tokens (BSD).

## [wmemchr\(\)](#)

## Description

Find character in memory, forward.

## Prototype

```
wchar_t *wmemchr(const wchar_t * s,  
                 wchar_t      c,  
                 size_t       n);
```

## Parameters

Parameter	Description
s	Pointer to object to search.
c	Wide character to search for.
n	Number of wide characters in object to search.

## Return value

- = NULL c does not occur in the object.
- ≠ NULL Pointer to the located wide character.

## Additional information

Locates the first occurrence of c in the initial n wide characters of the object pointed to by s. Unlike [wcschr\(\)](#), [wmemchr\(\)](#) does not terminate a search when a null wide character is found in the object pointed to by s.

## [wcschr\(\)](#)

## Description

Find character within string, forward.

## Prototype

```
wchar_t *wcschr(const wchar_t * s,  
                wchar_t      c);
```

## Parameters

Parameter	Description
s	Wide string to search.
c	Wide character to search for.

## Return value

Returns a pointer to the located wide character, or a null pointer if c does not occur in the wide string.



## Additional information

Locates the first occurrence of *c* in the wide string pointed to by *s*. The terminating wide null character is considered to be part of the string.

### *wcsnchr()*

## Description

Find character within string, forward, limit length.

## Prototype

```
wchar_t *wcsnchr(const wchar_t * s,  
                 size_t      n,  
                 wchar_t      c);
```

## Parameters

Parameter	Description
<i>s</i>	Pointer to wide string to search.
<i>n</i>	Number of wide characters to search.
<i>c</i>	Wide character to search for.

## Return value

Returns a pointer to the located wide character, or a null pointer if *c* does not occur in the string.

## Additional information

Searches not more than *n* wide characters to locate the first occurrence of *c* in the wide string pointed to by *s*. The terminating wide null character is considered to be part of the wide string.

### *wcsrchr()*

## Description

Find character within string, reverse.

## Prototype

```
wchar_t *wcsrchr(const wchar_t * s,  
                 wchar_t      c);
```

## Parameters

Parameter	Description
-----------	-------------

s            Pointer to wide string to search.  
c            Wide character to search for.

#### Return value

Returns a pointer to the located wide character, or a null pointer if c does not occur in the string.

#### Additional information

Locates the last occurrence of c in the wide string pointed to by s. The terminating wide null character is considered to be part of the string.

#### *wcslen()*

##### Description

Calculate length of string.

##### Prototype

```
size_t wcslen(const wchar_t * s);
```

##### Parameters

Parameter	Description
s	Pointer to zero-terminated wide string.

#### Return value

Returns the length of the wide string pointed to by s, that is the number of wide characters that precede the terminating wide null character.

#### *wcsnlen()*

##### Description

Calculate length of string, limit length (POSIX.1).

##### Prototype

```
size_t wcsnlen(const wchar_t * s,  
               size_t      n);
```

##### Parameters

Parameter	Description
s	Pointer to wide string.

n            Maximum number of wide characters to examine.

#### Return value

Returns the length of the wide string pointed to by s, up to a maximum of n wide characters. [wcsnlen\(\)](#) only examines the first n wide characters of the string s.

#### Notes

Conforms to POSIX.1-2008.

#### [wcsstr\(\)](#)

#### Description

Find string within string, forward.

#### Prototype

```
wchar_t *wcsstr(const wchar_t * s1,  
                const wchar_t * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string to search for.

#### Return value

Returns a pointer to the located wide string, or a null pointer if the wide string is not found. If s2 points to a wide string with zero length, [wcsstr\(\)](#) returns s1.

#### Additional information

Locates the first occurrence in the wide string pointed to by s1 of the sequence of wide characters (excluding the terminating null wide character) in the wide string pointed to by s2.

#### [wcsnstr\(\)](#)

#### Description

Find string within string, forward, limit length (BSD).

#### Prototype

```
wchar_t *wcsnstr(const wchar_t * s1,  
                const wchar_t * s2,  
                size_t n);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string to search for.
n	Maximum number of characters to search for.

#### Return value

Returns a pointer to the located wide string, or a null pointer if the wide string is not found. If s2 points to a wide string with zero length, [wcsnstr\(\)](#) returns s1.

#### Additional information

Searches at most n wide characters to locate the first occurrence in the wide string pointed to by s1 of the sequence of wide characters (excluding the terminating wide null character) in the string pointed to by s2.

#### Notes

Commonly found in Linux and BSD C libraries.

#### [wcspbrk\(\)](#)

#### Description

Find first occurrence of characters within string.

#### Prototype

```
wchar_t *wcspbrk(const wchar_t * s1,  
                const wchar_t * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string to search.
s2	Pointer to wide string to search for.

#### Return value

Returns a pointer to the first wide character, or a null pointer if no wide character from s2 occurs in s1.

## Additional information

Locates the first occurrence in the wide string pointed to by s1 of any wide character from the string pointed to by s2.

### *wcsspn()*

#### Description

Compute size of string prefixed by a set of characters.

#### Prototype

```
size_t wcsspn(const wchar_t * s1,  
              const wchar_t * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to zero-terminated wide string to search.
s2	Pointer to zero-terminated acceptable-set wide string.

#### Return value

Returns the length of the wide string pointed to by s1 which consists entirely of wide characters from the wide string pointed to by s2

## Additional information

Computes the length of the maximum initial segment of the wide string pointed to by s1 which consists entirely of wide characters from the string pointed to by s2.

### *wcscspn()*

#### Description

Compute size of string not prefixed by a set of characters.

#### Prototype

```
size_t wcscspn(const wchar_t * s1,  
               const wchar_t * s2);
```

#### Parameters

Parameter	Description
s1	Pointer to wide string to search.

s2            Pointer to wide string containing characters to skip.

#### Return value

Returns the length of the segment of wide string s1 prefixed by wide characters from s2.

#### Additional information

Computes the length of the maximum initial segment of the wide string pointed to by s1 which consists entirely of wide characters not from the wide string pointed to by s2.

#### [wcstok\(\)](#)

#### Description

Break string into tokens.

#### Prototype

```
wchar_t *wcstok(    wchar_t * s1,  
                   const wchar_t * s2,  
                   wchar_t ** ptr);
```

#### Parameters

Parameter	Description
s1	Pointer to zero-terminated wide string to parse.
s2	Pointer to zero-terminated set of separators.
ptr	Pointer to object that maintains parse state.

#### Return value

NULL if no further tokens else a pointer to the next token.

#### Additional information

A sequence of calls to [wcstok\(\)](#) breaks the wide string pointed to by s1 into a sequence of tokens, each of which is delimited by a wide character from the wide string pointed to by s2. The first call in the sequence has a non-null first argument; subsequent calls in the sequence have a null first argument. The separator wide string pointed to by s2 may be different from call to call.

The first call in the sequence searches the wide string pointed to by s1 for the wide first character that is not contained in the current separator wide string pointed to by s2. If no such wide character is found, then there are no tokens in the string pointed to by s1 and [wcstok\(\)](#) returns a null pointer. If such a wide character is found, it is the start of the first token.

`wcstok()` then searches from there for a wide character that is contained in the current separator wide string. If no such wide character is found, the current token extends to the end of the wide string pointed to by `s1`, and subsequent searches for a token will return a null pointer. If such a wide character is found, it is overwritten by a null wide character, which terminates the current token. `wcstok()` saves a pointer to the following wide character, from which the next search for a token will start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

See also

`wcssep()`.

`wcssep()`

Description

Break string into tokens (BSD).

Prototype

```
wchar_t *wcssep(      wchar_t ** stringp,  
                  const wchar_t * delim);
```

Parameters

Parameter	Description
<code>stringp</code>	Pointer to pointer to zero-terminated wide string.
<code>delim</code>	Pointer to delimiter set wide string.

Return value

See below.

Additional information

Locates, in the wide string referenced by `*stringp`, the first occurrence of any wide character in the wide string `delim` (or the terminating null character) and replaces it with a null wide character. The location of the next wide character after the delimiter wide character (or NULL, if the end of the wide string was reached) is stored in `*stringp`. The original value of `*stringp` is returned.

An empty field (that is, a wide character in the string `delim` occurs as the first character of `*stringp`) can be detected by comparing the location referenced by the returned pointer to the null wide character.

If `*stringp` is initially null, `wcssep()` returns null.

## Notes

Commonly found in Linux and BSD C libraries.

### Multi-byte/wide string conversion functions

Function	Description
<a href="#">mbsinit()</a>	Query initial conversion state.
<a href="#">mbrlen()</a>	Count number of bytes in multi-byte character, restartable.
<a href="#">mbrlen_l()</a>	Count number of bytes in multi-byte character, restartable, per locale (POSIX.1).
<a href="#">mbrtowc()</a>	Convert multi-byte character to wide character, restartable.
<a href="#">mbrtowc_l()</a>	Convert multi-byte character to wide character, restartable, per locale (POSIX.1).
<a href="#">wctob()</a>	Convert wide character to single-byte character.
<a href="#">wctob_l()</a>	Convert wide character to single-byte character, per locale (POSIX.1).
<a href="#">wrtomb()</a>	Convert wide character to multi-byte character, restartable.
<a href="#">wrtomb_l()</a>	Convert wide character to multi-byte character, restartable, per locale (POSIX.1).
<a href="#">wcsrtombs()</a>	Convert wide string to multi-byte string, restartable.
<a href="#">wcsrtombs_l()</a>	Convert wide string to multi-byte string, restartable (POSIX.1).

#### [mbsinit\(\)](#)

##### Description

Query initial conversion state.

##### Prototype

```
int mbsinit(const mbstate_t * ps);
```

##### Parameters

Parameter	Description
ps	Pointer to conversion state.

##### Return value

Returns nonzero (true) if ps is a null pointer or if the pointed-to object describes an initial conversion state; otherwise, returns zero.

#### [mbrlen\(\)](#)

##### Description

Count number of bytes in multi-byte character, restartable.



## Prototype

```
size_t mbrlen(const char * s,  
              size_t n,  
              mbstate_t * ps);
```

## Parameters

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.
ps	Pointer to multi-byte conversion state.

## Return value

Number of bytes in multi-byte character.

## Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the current locale.

Except that except that the expression designated by ps is evaluated only once, this function is equivalent to the call:

```
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

where internal is the mbstate\_t object for the [mbrlen\(\)](#) function.

See also

[mbrlen\\_l\(\)](#), [mbrtowc\(\)](#)

[mbrlen\\_l\(\)](#)

## Description

Count number of bytes in multi-byte character, restartable, per locale (POSIX.1).

## Prototype

```
size_t mbrlen_l(const char * s,  
               size_t n,  
               mbstate_t * ps,  
               locale_t loc);
```

## Parameters

---

Parameter	Description
s	Pointer to multi-byte character.
n	Maximum number of bytes to examine.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

Return value

Number of bytes in multi-byte character.

Additional information

Determines the number of bytes contained in the multi-byte character pointed to by s in the locale loc.

Except that except that the expression designated by ps is evaluated only once, this function is equivalent to the call:

```
mbrtowc_1(NULL, s, n, ps != NULL ? ps : &internal, loc);
```

where internal is the mbstate\_t object for the [mbrlen\(\)](#) function,

Notes

Conforms to POSIX.1-2008.

See also

[mbrlen\\_l\(\)](#), [mbrtowc\(\)](#)

[mbrtowc\(\)](#)

Description

Convert multi-byte character to wide character, restartable.

Prototype

```
size_t mbrtowc(    wchar_t    * pwc,  
                  const char    * s,  
                  size_t        n,  
                  mbstate_t * ps);
```

Parameters

---

Parameter	Description
pwc	Pointer to object that receives the wide character.

---

s            Pointer to multi-byte character string.  
n            Maximum number of bytes that will be examined.  
ps           Pointer to multi-byte conversion state.

#### Return value

If s is a null pointer, [mbrtowc\(\)](#) is equivalent to `mbrtowc(NULL, "", 1, ps)`, ignoring pwc and n.

If s is not null and the object that s points to is a wide null character, [mbrtowc\(\)](#) returns 0.

If s is not null and the object that s points to forms a valid multi-byte character in the current locale with a most n bytes, [mbrtowc\(\)](#) returns the length in bytes of the multi-byte character and stores that wide character to the object pointed to by pwc (if pwc is not null).

If the object that s points to forms an incomplete, but possibly valid, multi-byte character, [mbrtowc\(\)](#) returns -2.

If the object that s points to does not form a partial multi-byte character, [mbrtowc\(\)](#) returns -1.

#### Additional information

Converts a single multi-byte character to a wide character in the current locale.

See also

[mbtowc\(\)](#), [mbrtowc\\_l\(\)](#)

[mbrtowc\\_l\(\)](#)

#### Description

Convert multi-byte character to wide character, restartable, per locale (POSIX.1).

#### Prototype

```
size_t mbrtowc_l(      wchar_t    * pwc,  
                      const char  * s,  
                      size_t      n,  
                      mbstate_t * ps,  
                      locale_t loc);
```

#### Parameters

Parameter	Description
pwc	Pointer to object that receives the wide character.
s	Pointer to multi-byte character string.
n	Maximum number of bytes that will be examined.

ps            Pointer to multi-byte conversion state.  
loc           Locale used for conversion.

#### Return value

If *s* is a null pointer, `mbrtowc()` is equivalent to `mbrtowc(NULL, "", 1, ps)`, ignoring *pwc* and *n*.

If *s* is not null and the object that *s* points to is a wide null character, `mbrtowc()` returns 0.

If *s* is not null and the object that *s* points to forms a valid multi-byte character in the locale *loc* with a most *n* bytes, `mbrtowc()` returns the length in bytes of the multi-byte character and stores that wide character to the object pointed to by *pwc* (if *pwc* is not null).

If the object that *s* points to forms an incomplete, but possibly valid, multi-byte character, `mbrtowc()` returns -2.

If the object that *s* points to does not form a partial multi-byte character, `mbrtowc()` returns -1.

#### Additional information

Converts a single multi-byte character to a wide character in the locale *loc*.

#### Notes

Conforms to POSIX.1-2008.

See also

[mbtowc\(\)](#), [mbrtowc\\_l\(\)](#)

[wctob\(\)](#)

#### Description

Convert wide character to single-byte character.

#### Prototype

```
int wctob(wint_t c);
```

#### Parameters

Parameter	Description
<i>c</i>	Character to convert.

#### Return value

Returns EOF if *c* does not correspond to a multi-byte character with length one in the initial shift state in the current locale. Otherwise, it returns the single-byte representation of that character as an unsigned char converted to an int.

#### Additional information

Determines whether *c* corresponds to a member of the extended character set whose multi-byte character representation is a single byte in the current locale when in the initial shift state.

#### *wctob\_l()*

##### Description

Convert wide character to single-byte character, per locale (POSIX.1).

##### Prototype

```
int wctob_l(wint_t c,  
            locale_t loc);
```

##### Parameters

Parameter	Description
<i>c</i>	Character to convert.
<i>loc</i>	Locale used for conversion.

##### Return value

Returns EOF if *c* does not correspond to a multi-byte character with length one in the initial shift state in the locale *loc*. Otherwise, it returns the single-byte representation of that character as an unsigned char converted to an int.

#### Additional information

Determines whether *c* corresponds to a member of the extended character set whose multi-byte character representation is a single byte in the locale *loc* when in the initial shift state.

#### *wcrtomb()*

##### Description

Convert wide character to multi-byte character, restartable.

##### Prototype

```
size_t wcrtomb(char * s,  
               wchar_t wc,  
               mbstate_t * ps);
```

## Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte character.
wc	Wide character to convert.
ps	Pointer to multi-byte conversion state.

## Return value

Returns the number of bytes stored in the array object. When wc is not a valid wide character, an encoding error occurs: `wcrtomb()` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1)`; the conversion state is unspecified.

## Additional information

If s is a null pointer, `wcrtomb()` is equivalent to the call `wcrtomb(buf, 0, ps)` where buf is an internal buffer.

If s is not a null pointer, `wcrtomb()` determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by wc in the locale loc, and stores the multi-byte character representation in the array whose first element is pointed to by s. At most `MB_CUR_MAX` bytes are stored. If wc is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

## `wcrtomb_l()`

## Description

Convert wide character to multi-byte character, restartable, per locale (POSIX.1).

## Prototype

```
size_t wcrtomb_l(char      * s,  
                  wchar_t wc,  
                  mbstate_t * ps,  
                  locale_t loc);
```

## Parameters

Parameter	Description
s	Pointer to array that receives the multi-byte character.
wc	Wide character to convert.
ps	Pointer to multi-byte conversion state.
loc	Locale used for conversion.

## Return value

Returns the number of bytes stored in the array object. When `wc` is not a valid wide character, an encoding error occurs: `wcrtomb_l()` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1)`; the conversion state is unspecified.

## Additional information

If `s` is a null pointer, `wcrtomb()` is equivalent to the call `wcrtomb(buf, 0, ps)` where `buf` is an internal buffer.

If `s` is not a null pointer, `wcrtomb()` determines the number of bytes needed to represent the multi-byte character that corresponds to the wide character given by `wc` in the current locale, and stores the multi-byte character representation in the array whose first element is pointed to by `s`. At most `MB_CUR_MAX` bytes are stored. If `wc` is a null wide character, a null byte is stored; the resulting state described is the initial conversion state.

## `wcsrtombs()`

## Description

Convert wide string to multi-byte string, restartable.

## Prototype

```
size_t wcsrtombs(    char        * dst,
                    const wchar_t ** src,
                    size_t      len,
                    mbstate_t   * ps);
```

## Parameters

Parameter	Description
<code>dst</code>	Pointer to array that receives the multi-byte string.
<code>src</code>	Indirect pointer to wide character string being converted.
<code>len</code>	Maximum number of bytes to write into the array pointed to by <code>dst</code> .
<code>ps</code>	Pointer to multi-byte conversion state.

## Return value

If conversion stops because a wide character is reached that does not correspond to a valid multi-byte character, an encoding error occurs: `wcsrtombs()` stores the value of the macro `EILSEQ` in `errno` and returns `(size_t)(-1)`; the conversion state is unspecified. Otherwise, it returns the number of bytes in the resulting multi-byte character sequence, not including the terminating null character (if any).

## Additional information

Converts a sequence of wide characters in the current locale from the array indirectly pointed to by `src` into a sequence of corresponding multi-byte characters that begins in the conversion state described by the object pointed to by `ps`. If `dst` is not a null pointer, the converted characters are then stored into the array pointed to by `dst`. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases: when a wide character is reached that does not correspond to a valid multi-byte character, or (if `dst` is not a null pointer) when the next multi-byte character would exceed the limit of `len` total bytes to be stored into the array pointed to by `dst`. Each conversion takes place as if by a call to [wcrctomb\(\)](#).

If `dst` is not a null pointer, the pointer object pointed to by `src` is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described is the initial conversion state.

## [wcsrtombs\\_l\(\)](#)

### Description

Convert wide string to multi-byte string, restartable (POSIX.1).

### Prototype

```
size_t wcsrtombs_l(    char      * dst,
                      const wchar_t ** src,
                      size_t      len,
                      mbstate_t   * ps,
                      locale_t     loc);
```

### Parameters

Parameter	Description
<code>dst</code>	Pointer to array that receives the multi-byte string.
<code>src</code>	Indirect pointer to wide character string being converted.
<code>len</code>	Maximum number of bytes to write into the array pointed to by <code>dst</code> .
<code>ps</code>	Pointer to multi-byte conversion state.
<code>loc</code>	Locale used for conversion.

### Return value

If conversion stops because a wide character is reached that does not correspond to a valid multi-byte character, an encoding error occurs: [wcsrtombs\(\)](#) stores the value of the macro `EILSEQ` in



errno and returns (size\_t)(-1); the conversion state is unspecified. Otherwise, it returns the number of bytes in the resulting multi-byte character sequence, not including the terminating null character (if any).

#### Additional information

Converts a sequence of wide characters in the locale loc from the array indirectly pointed to by src into a sequence of corresponding multi-byte characters that begins in the conversion state described by the object pointed to by ps. If dst is not a null pointer, the converted characters are then stored into the array pointed to by dst. Conversion continues up to and including a terminating null wide character, which is also stored.

Conversion stops earlier in two cases: when a wide character is reached that does not correspond to a valid multi-byte character, or (if dst is not a null pointer) when the next multi-byte character would exceed the limit of len total bytes to be stored into the array pointed to by dst. Each conversion takes place as if by a call to [wctomb\\_l\(\)](#).

If dst is not a null pointer, the pointer object pointed to by src is assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described is the initial conversion state.

#### Notes

Conforms to POSIX.1-2008.

### <wctype.h>

#### Classification functions

Function	Description
<a href="#">iswcntrl()</a>	Is character a control?
<a href="#">iswcntrl_l()</a>	Is character a control, per locale? (POSIX.1).
<a href="#">iswblank()</a>	Is character a blank?
<a href="#">iswblank_l()</a>	Is character a blank, per locale? (POSIX.1).
<a href="#">iswspace()</a>	Is character a whitespace character?
<a href="#">iswspace_l()</a>	Is character a whitespace character, per locale? (POSIX.1).
<a href="#">iswpunct()</a>	Is character a punctuation mark?
<a href="#">iswpunct_l()</a>	Is character a punctuation mark, per locale? (POSIX.1).
<a href="#">iswdigit()</a>	Is character a decimal digit?
<a href="#">iswdigit_l()</a>	Is character a decimal digit, per locale? (POSIX.1).
<a href="#">iswxdigit()</a>	Is character a hexadecimal digit?

<code>iswxdigit_l()</code>	Is character a hexadecimal digit, per locale? (POSIX.1).
<code>iswalpha()</code>	Is character alphabetic?
<code>iswalpha_l()</code>	Is character alphabetic, per locale? (POSIX.1).
<code>iswalnum()</code>	Is character alphanumeric?
<code>iswalnum_l()</code>	Is character alphanumeric, per locale? (POSIX.1).
<code>iswupper()</code>	Is character an uppercase letter?
<code>iswupper_l()</code>	Is character an uppercase letter, per locale? (POSIX.1).
<code>iswlower()</code>	Is character a lowercase letter?
<code>iswlower_l()</code>	Is character a lowercase letter, per locale? (POSIX.1).
<code>iswprint()</code>	Is character printable?
<code>iswprint_l()</code>	Is character printable, per locale? (POSIX.1).
<code>iswgraph()</code>	Is character any printing character?
<code>iswgraph_l()</code>	Is character any printing character, per locale? (POSIX.1).
<code>iswctype()</code>	Construct character mapping.
<code>iswctype_l()</code>	Construct character mapping, per locale (POSIX.1).
<code>wctype()</code>	Construct character class.

### *iswcntrl()*

#### Description

Is character a control?

#### Prototype

```
int iswcntrl(wint_t c);
```

#### Parameters

Parameter	Description
<code>c</code>	Wide character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument `c` is a control character in the current locale.

### *iswcntrl\_l()*

#### Description

Is character a control, per locale? (POSIX.1).

## Prototype

```
int iswcntrl_l(wint_t c,  
               locale_t loc);
```

## Parameters

Parameter	Description
<code>c</code>	Wide character to test.
<code>loc</code>	Locale used to test <code>c</code> .

## Return value

Returns nonzero (true) if and only if the value of the argument `c` is a control character in the locale `loc`.

## Notes

Conforms to POSIX.1-2017.

[\*iswblank\(\)\*](#)

## Description

Is character a blank?

## Prototype

```
int iswblank(wint_t c);
```

## Parameters

Parameter	Description
<code>c</code>	Wide character to test.

## Return value

Returns nonzero (true) if and only if the value of the argument `c` is either a space character or tab character in the current locale.

[\*iswblank\\_l\(\)\*](#)

## Description

Is character a blank, per locale? (POSIX.1).

## Prototype

```
int iswblank_l(wint_t c,
               locale_t loc);
```

#### Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is either a space character or the tab character in locale loc.

#### Notes

Conforms to POSIX.1-2017.

#### [iswspace\(\)](#)

##### Description

Is character a whitespace character?

##### Prototype

```
int iswspace(wint_t c);
```

#### Parameters

Parameter	Description
c	Wide character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the current locale. The standard white-space characters are space, form feed, new-line, carriage return, horizontal tab, and vertical tab.

#### [iswspace\\_l\(\)](#)

##### Description

Is character a whitespace character, per locale? (POSIX.1).

##### Prototype

```
int iswspace_l(wint_t c,  
               locale_t loc);
```

#### Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is a standard white-space character in the locale loc.

#### Notes

Conforms to POSIX.1-2017.

#### *iswpunct()*

##### Description

Is character a punctuation mark?

##### Prototype

```
int iswpunct(wint_t c);
```

#### Parameters

Parameter	Description
c	Wide character to test.

#### Return value

Returns nonzero (true) for every printing character for which neither *isspace()* nor *isalnum()* is true in the current locale.

#### *iswpunct\_l()*

##### Description

Is character a punctuation mark, per locale? (POSIX.1).

##### Prototype

```
int iswpunct_l(wint_t c,  
               locale_t loc);
```

## Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

## Return value

Returns nonzero (true) for every printing character for which neither `isspace()` nor `isalnum()` is true in the locale loc.

## Notes

Conforms to POSIX.1-2017.

## `iswdigit()`

### Description

Is character a decimal digit?

### Prototype

```
int iswdigit(wint_t c);
```

## Parameters

Parameter	Description
c	Wide character to test.

## Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the current locale.

## `iswdigit_l()`

### Description

Is character a decimal digit, per locale? (POSIX.1)

### Prototype

```
int iswdigit_l(wint_t c,  
               locale_t loc);
```

## Parameters

Parameter	Description
-----------	-------------

c	Wide character to test.
loc	Locale used to test c.

Return value

Returns nonzero (true) if and only if the value of the argument c is a digit in the locale loc.

Notes

Conforms to POSIX.1-2017.

*iswxdigit()*

Description

Is character a hexadecimal digit?

Prototype

```
int iswxdigit(wint_t c);
```

Parameters

Parameter	Description
c	Wide character to test.

Return value

Returns nonzero (true) if and only if the value of the argument c is a hexadecimal digit in the current locale.

*iswxdigit\_l()*

Description

Is character a hexadecimal digit, per locale? (POSIX.1).

Prototype

```
int iswxdigit_l(wint_t c,  
                locale_t loc);
```

Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

## Return value

Returns nonzero (true) if and only if the value of the argument `c` is a hexadecimal digit in the current locale.

## Notes

Conforms to POSIX.1-2017.

## *iswalpha()*

### Description

Is character alphabetic?

### Prototype

```
int iswalpha(wint_t c);
```

### Parameters

Parameter	Description
<code>c</code>	Wide character to test.

## Return value

Returns true if the character `c` is alphabetic in the current locale. That is, any character for which [iswupper\(\)](#) or [iswlower\(\)](#) returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of [iswcntrl\(\)](#), [iswdigit\(\)](#), [iswpunct\(\)](#), or [isspace\(\)](#) is true.

In the C locale, [isalpha\(\)](#) returns nonzero (true) if and only if [isupper\(\)](#) or [islower\(\)](#) return true for value of the argument `c`.

## *iswalpha\_l()*

### Description

Is character alphabetic, per locale? (POSIX.1).

### Prototype

```
int iswalpha_l(wint_t c,  
               locale_t loc);
```

### Parameters

Parameter	Description
-----------	-------------



c	Wide character to test.
loc	Locale used to test c.

#### Return value

Returns true if the wide character c is alphabetic in the locale loc. That is, any character for which `iswupper()` or `iswlower()` returns true is considered alphabetic in addition to any of the locale-specific set of alphabetic characters for which none of `iswcntrl_l()`, `iswdigit_l()`, `iswpunct_l()`, or `iswspace_l()` is true in the locale loc.

In the C locale, `iswalphabet_l()` returns nonzero (true) if and only if `iswupper_l()` or `iswlower_l()` return true for value of the argument c.

#### Notes

Conforms to POSIX.1-2017.

#### `iswalnum()`

##### Description

Is character alphanumeric?

##### Prototype

```
int iswalnum(wint_t c);
```

##### Parameters

Parameter	Description
c	Wide character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is an alphabetic or numeric character in the current locale.

#### `iswalnum_l()`

##### Description

Is character alphanumeric, per locale? (POSIX.1).

##### Prototype

```
int iswalnum_l(wint_t c,  
               locale_t loc);
```

## Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

## Return value

Returns nonzero (true) if and only if the value of the argument c is an alphabetic or numeric character in the locale loc.

## Notes

Conforms to POSIX.1-2017.

## *iswupper()*

### Description

Is character an uppercase letter?

### Prototype

```
int iswupper(wint_t c);
```

## Parameters

Parameter	Description
c	Wide character to test.

## Return value

Returns nonzero (true) if and only if the value of the argument c is an uppercase letter in the current locale.

## *iswupper\_l()*

### Description

Is character an uppercase letter, per locale? (POSIX.1).

### Prototype

```
int iswupper_l(wint_t c,  
               locale_t loc);
```

## Parameters

---

Parameter	Description
<code>c</code>	Wide character to test.
<code>loc</code>	Locale used to test <code>c</code> .

#### Return value

Returns nonzero (true) if and only if the value of the argument `c` is an uppercase letter in the locale `loc`.

#### Notes

Conforms to POSIX.1-2017.

#### *iswlower()*

##### Description

Is character a lowercase letter?

##### Prototype

```
int iswlower(wint_t c);
```

##### Parameters

---

Parameter	Description
<code>c</code>	Wide character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument `c` is a lowercase letter in the current locale.

#### *iswlower\_l()*

##### Description

Is character a lowercase letter, per locale? (POSIX.1).

##### Prototype

```
int iswlower_l(wint_t c,  
               locale_t loc);
```

##### Parameters

---

Parameter	Description
-----------	-------------

---

c	Wide character to test.
loc	Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is a lowercase letter in the locale loc.

#### Notes

Conforms to POSIX.1-2017.

#### *iswprint()*

##### Description

Is character printable?

##### Prototype

```
int iswprint(wint_t c);
```

##### Parameters

Parameter	Description
c	Wide character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the current locale.

#### *iswprint\_l()*

##### Description

Is character printable, per locale? (POSIX.1).

##### Prototype

```
int iswprint_l(wint_t c,  
               locale_t loc);
```

##### Parameters

Parameter	Description
c	Wide character to test.

loc            Locale used to test c.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character including space in the locale loc.

#### Notes

Conforms to POSIX.1-2017.

#### *iswgraph()*

##### Description

Is character any printing character?

##### Prototype

```
int iswgraph(wint_t c);
```

##### Parameters

Parameter	Description
c	Wide character to test.

#### Return value

Returns nonzero (true) if and only if the value of the argument c is any printing character except space in the current locale.

#### *iswgraph\_l()*

##### Description

Is character any printing character, per locale? (POSIX.1).

##### Prototype

```
int iswgraph_l(wint_t c,  
               locale_t loc);
```

##### Parameters

Parameter	Description
c	Wide character to test.
loc	Locale used to test c.

## Return value

Returns nonzero (true) if and only if the value of the argument `c` is any printing character except space in the locale `loc`.

## Notes

Conforms to POSIX.1-2017.

## *iswctype()*

## Description

Construct character mapping.

## Prototype

```
int iswctype(wint_t c,
             wctype_t t);
```

## Parameters

Parameter	Description
<code>c</code>	Wide character to test.
<code>t</code>	Property to test, typically delivered by calling <code>wctype()</code> .

## Return value

Returns nonzero (true) if and only if the wide character `c` has the property `t` in the current locale.

## Additional information

Determines whether the wide character `c` has the property described by `t` in the current locale.

## *iswctype\_l()*

## Description

Construct character mapping, per locale (POSIX.1).

## Prototype

```
int iswctype_l(wint_t c,
               wctype_t t,
               locale_t loc);
```

## Parameters

Parameter	Description
-----------	-------------

---

c	Wide character to test.
t	Property to test, typically delivered by calling <code>wctype_l()</code> .
loc	Locale used for mapping.

#### Return value

Returns nonzero (true) if and only if the wide character `c` has the property `t` in the locale `loc`.

#### Additional information

Determines whether the wide character `c` has the property described by `t` in the locale `loc`.

#### Notes

Conforms to POSIX.1-2017.

#### `wctype()`

#### Description

Construct character class.

#### Prototype

```
wctype_t wctype(char const * name);
```

#### Parameters

Parameter	Description
name	Name of mapping.

#### Return value

Character class; zero if class unrecognized.

#### Additional information

Constructs a value of type `wctype_t` that describes a class of wide characters identified by the string argument property.

If property identifies a valid class of wide characters in the current locale, returns a nonzero value that is valid as the second argument to `iswctype()`; otherwise, it returns zero.

#### Notes

The only mappings supported are:

- “alnum”

- “alpha”,
- “blank”
- “cntrl”
- “digit”
- “graph”
- “lower”
- “print”
- “punct”
- “space”
- “upper”
- “xdigit”

### Conversion functions

Function	Description
<a href="#">towupper()</a>	Convert uppercase character to lowercase.
<a href="#">towupper_l()</a>	Convert uppercase character to lowercase, per locale (POSIX.1).
<a href="#">tolower()</a>	Convert uppercase character to lowercase.
<a href="#">tolower_l()</a>	Convert uppercase character to lowercase, per locale (POSIX.1).
<a href="#">towctrans()</a>	Translate character.
<a href="#">towctrans_l()</a>	Translate character, per locale (POSIX.1).
<a href="#">wctrans()</a>	Construct character mapping.
<a href="#">wctrans_l()</a>	Construct character mapping, per locale (POSIX.1).

#### [towupper\(\)](#)

##### Description

Convert uppercase character to lowercase.

##### Prototype

```
wint_t towupper(wint_t c);
```

##### Parameters

Parameter	Description
c	Wide character to convert.

##### Return value

Converted wide character.



## Additional information

Converts a lowercase letter to a corresponding uppercase letter.

If the argument *c* is a wide character for which `iswlower()` is true and there are one or more corresponding wide characters, in the current current locale, for which `iswupper()` is true, `towupper()` returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, *c* is returned unchanged.

### `towupper_l()`

#### Description

Convert uppercase character to lowercase, per locale (POSIX.1).

#### Prototype

```
wint_t towupper_l(wint_t c,  
                  locale_t loc);
```

#### Parameters

Parameter	Description
<i>c</i>	Wide character to convert.
<i>loc</i>	Locale used to convert <i>c</i> .

#### Return value

Converted wide character.

## Additional information

Converts a lowercase letter to a corresponding uppercase letter.

If the argument *c* is a wide character for which `iswlower_l()` is true and there are one or more corresponding wide characters, in the current locale *loc*, for which `iswupper_l()` is true, `towupper_l()` returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, *c* is returned unchanged.

#### Notes

Conforms to POSIX.1-2017.

### `towlower()`

#### Description

Convert uppercase character to lowercase.

## Prototype

```
wint_t tolower(wint_t c);
```

## Parameters

Parameter	Description
<code>c</code>	Wide character to convert.

## Return value

Converted wide character.

## Additional information

Converts an uppercase letter to a corresponding lowercase letter.

If the argument `c` is a wide character for which `iswupper()` is true and there are one or more corresponding wide characters, in the current locale, for which `iswlower()` is true, `tolower()` returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, `c` is returned unchanged.

## `tolower_l()`

## Description

Convert uppercase character to lowercase, per locale (POSIX.1).

## Prototype

```
wint_t tolower_l(wint_t c,  
                 locale_t loc);
```

## Parameters

Parameter	Description
<code>c</code>	Wide character to convert.
<code>loc</code>	Locale used to convert <code>c</code> .

## Return value

Converted wide character.

## Additional information

Converts an uppercase letter to a corresponding lowercase letter.

If the argument `c` is a wide character for which `iswupper_l()` is true and there are one or more corresponding wide characters, in the locale `loc`, for which `iswlower_l()` is true, `towlower_l()` returns one (and always the same one for any given locale) of the corresponding wide characters; otherwise, `c` is returned unchanged.

#### Notes

Conforms to POSIX.1-2017.

#### `towctrans()`

##### Description

Translate character.

##### Prototype

```
wint_t towctrans(wint_t c,  
                 wctrans_t t);
```

##### Parameters

Parameter	Description
<code>c</code>	Wide character to convert.
<code>t</code>	Mapping to use for conversion.

##### Return value

Converted wide character.

##### Additional information

Maps the wide character `c` using the mapping described by `t` in the current locale.

#### `towctrans_l()`

##### Description

Translate character, per locale (POSIX.1).

##### Prototype

```
wint_t towctrans_l(wint_t c,  
                   wctrans_t t,  
                   locale_t loc);
```

##### Parameters

---

Parameter	Description
c	Wide character to convert.
t	Mapping to use for conversion.
loc	Locale used for conversion.

Return value

Converted wide character.

Additional information

Maps the wide character c using the mapping described by t in the locale loc.

Notes

Conforms to POSIX.1-2017.

[wctrans\(\)](#)

Description

Construct character mapping.

Prototype

```
wctrans_t wctrans(const char * name);
```

Parameters

---

Parameter	Description
name	Name of mapping.

Return value

Transformation mapping; zero if mapping unrecognized.

Additional information

Constructs a value of type wctrans\_t that describes a mapping between wide characters identified by the string argument property.

If property identifies a valid mapping of wide characters in the current locale, [wctrans\\_l\(\)](#) returns a nonzero value that is valid as the second argument to [towctrans\(\)](#); otherwise, it returns zero.

The only mappings supported are “tolower” and “toupper”.

## `wctrans_l()`

### Description

Construct character mapping, per locale (POSIX.1).

### Prototype

```
wctrans_t wctrans_l(const char * name,  
                    locale_t loc);
```

### Parameters

Parameter	Description
name	Name of mapping.
loc	Locale used for mapping.

### Return value

Transformation mapping; zero if mapping unrecognized.

### Additional information

Constructs a value of type `wctrans_t` that describes a mapping between wide characters identified by the string argument property.

If property identifies a valid mapping of wide characters in the locale `loc`, `wctrans_l()` returns a nonzero value that is valid as the second argument to `towctrans()`; otherwise, it returns zero.

The only mappings supported are “tolower” and “toupper”.

### Notes

Conforms to POSIX.1-2017.

## <locale.h>

### Locale management

Function	Description
<code>newlocale()</code>	Duplicate locale data (POSIX.1).
<code>duplocale()</code>	Duplicate locale data (POSIX.1).
<code>freelocale()</code>	Free locale (POSIX.1).
<code>localeconv_l()</code>	Get locale data (POSIX.1).

## *newlocale()*

### Description

Duplicate locale data (POSIX.1).

### Prototype

```
locale_t newlocale(      int      category_mask,
                        const char * loc,
                        locale_t  base);
```

### Parameters

Parameter	Description
category_mask	Locale categories to be set or modified.
loc	Locale name.
base	Base to modify or NULL to create a new locale.

### Return value

Pointer to modified locale.

### Additional information

Creates a new locale object or modifies an existing one. If the base argument is NULL, a new locale object is created.

category\_mask specifies the locale categories to be set or modified. Values for category\_mask are constructed by a bitwise-inclusive OR of the symbolic constants LC\_CTYPE\_MASK, LC\_NUMERIC\_MASK, LC\_TIME\_MASK, LC\_COLLATE\_MASK, LC\_MONETARY\_MASK, and LC\_MESSAGES\_MASK.

For each category with the corresponding bit set in category\_mask, the data from the locale named by loc is used. In the case of modifying an existing locale object, the data from the locale named by loc replaces the existing data within the locale object. If a completely new locale object is created, the data for all sections not requested by category\_mask are taken from the default locale.

The locales “C” and “POSIX” are equivalent and defined for all settings of category\_mask:

If loc is NULL, then the “C” locale is used. If loc is an empty string, [newlocale\(\)](#) will use the default locale.

If base is NULL, the current locale is used. If base is LC\_GLOBAL\_LOCALE, the global locale is used.

If mask is LC\_ALL\_MASK, base is ignored.

## Notes

Conforms to POSIX.1-2008.

POSIX.1-2008 does not specify whether the locale object pointed to by base is modified or whether it is freed and a new locale object created.

The category mask LC\_MESSAGES\_MASK is not implemented as POSIX messages are not implemented.

## *duplocale()*

### Description

Duplicate locale data (POSIX.1).

### Prototype

```
locale_t duplocale(locale_t base);
```

### Parameters

Parameter	Description
base	Locale to duplicate.

### Return value

If there is insufficient memory to duplicate loc, returns a NULL and sets errno to ENOMEM as required by POSIX.1-2008. Otherwise, returns a new, duplicated locale.

### Additional information

Duplicates the locale object referenced by loc. Duplicated locales must be freed with [freelocale\(\)](#).

## Notes

Conforms to POSIX.1-2008.

## *freelocale()*

### Description

Free locale (POSIX.1).

### Prototype

```
int freelocale(locale_t loc);
```

### Parameters

---

Parameter	Description
loc	Locale to free.

Return value

Zero on success, -1 on error.

Additional information

Frees the storage associated with loc.

Notes

Conforms to POSIX.1-2008.

[\*localeconv\\_l\(\)\*](#)

Description

Get locale data (POSIX.1).

Prototype

```
localeconv_l(locale_t loc);
```

Parameters

---

Parameter	Description
loc	Locale to inquire.

Return value

Returns a pointer to a structure of type lconv with the corresponding values for the locale loc filled in.

Notes

Conforms to POSIX.1-2008.

## Compiler support API

### GNU library API

#### Integer arithmetic

---

Function	Description
<code>__mulsi3</code>	Multiply, signed 32-bit integer.
<code>__muldi3</code>	Multiply, signed 64-bit integer.

---



<code>__divsi3</code>	Divide, signed 32-bit integer.
<code>__divdi3</code>	Divide, signed 64-bit integer.
<code>__udivsi3</code>	Divide, unsigned 32-bit integer.
<code>__udivdi3</code>	Divide, unsigned 64-bit integer.
<code>__modsi3</code>	Remainder after divide, signed 32-bit integer.
<code>__moddi3</code>	Remainder after divide, signed 64-bit integer.
<code>__umodsi3</code>	Remainder after divide, unsigned 32-bit integer.
<code>__umoddi3</code>	Remainder after divide, unsigned 64-bit integer.
<code>__udivmodsi4</code>	Divide with remainder, unsigned 32-bit integer.
<code>__udivmoddi4</code>	Divide with remainder, unsigned 64-bit integer.
<code>__clzsi2</code>	Count leading zeros, 32-bit integer.
<code>__clzdi2</code>	Count leading zeros, 64-bit integer.
<code>__popcountsi2</code>	Population count, 32-bit integer.
<code>__popcountdi2</code>	Population count, 64-bit integer.
<code>__paritysi2</code>	Parity, 32-bit integer.
<code>__paritydi2</code>	Parity, 64-bit integer.

### `__mulsi3()`

#### Description

Multiply, signed 32-bit integer.

#### Prototype

```
__SEGGER_RTL_U32 __mulsi3(__SEGGER_RTL_U32 a,
                          __SEGGER_RTL_U32 b);
```

#### Parameters

Parameter	Description
a	Multiplier.
b	Multiplicand.

#### Return value

Product.

### `__muldi3()`

#### Description

Multiply, signed 64-bit integer.

Prototype

```
__SEGGER_RTL_U64 __muldi3(__SEGGER_RTL_U64 a,  
                           __SEGGER_RTL_U64 b);
```

Parameters

Parameter	Description
a	Multiplier.
b	Multiplicand.

Return value

Product.

[\\_\\_divsi3\(\)](#)

Description

Divide, signed 32-bit integer.

Prototype

```
int32_t __divsi3(int32_t num,  
                 int32_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Quotient.

[\\_\\_divdi3\(\)](#)

Description

Divide, signed 64-bit integer.

Prototype

```
int64_t __divdi3(int64_t num,  
                 int64_t den);
```

## Parameters

Parameter	Description
num	Dividend.
den	Divisor.

## Return value

Quotient.

[\\_\\_udivsi3\(\)](#)

## Description

Divide, unsigned 32-bit integer.

## Prototype

```
__SEGGER_RTL_U32 __udivsi3(__SEGGER_RTL_U32 num,  
                           __SEGGER_RTL_U32 den);
```

## Parameters

Parameter	Description
num	Dividend.
den	Divisor.

## Return value

Quotient.

[\\_\\_udivdi3\(\)](#)

## Description

Divide, unsigned 64-bit integer.

## Prototype

```
__SEGGER_RTL_U64 __udivdi3(__SEGGER_RTL_U64 num,  
                           __SEGGER_RTL_U64 den);
```

## Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Quotient.

[`\_\_modsi3\(\)`](#)

Description

Remainder after divide, signed 32-bit integer.

Prototype

```
int32_t __modsi3(int32_t num,  
                int32_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Remainder.

[`\_\_moddi3\(\)`](#)

Description

Remainder after divide, signed 64-bit integer.

Prototype

```
int64_t __moddi3(int64_t num,  
                int64_t den);
```

Parameters

Parameter	Description
num	Dividend.
den	Divisor.

Return value

Remainder.

## [\\_\\_umodsi3\(\)](#)

### Description

Remainder after divide, unsigned 32-bit integer.

### Prototype

```
__SEGGER_RTL_U32 __umodsi3(__SEGGER_RTL_U32 num,  
                           __SEGGER_RTL_U32 den);
```

### Parameters

Parameter	Description
num	Dividend.
den	Divisor.

### Return value

Remainder.

## [\\_\\_umoddi3\(\)](#)

### Description

Remainder after divide, unsigned 64-bit integer.

### Prototype

```
__SEGGER_RTL_U64 __umoddi3(__SEGGER_RTL_U64 num,  
                           __SEGGER_RTL_U64 den);
```

### Parameters

Parameter	Description
num	Dividend.
den	Divisor.

### Return value

Remainder.

## [\\_\\_udivmodsi4\(\)](#)

### Description

Divide with remainder, unsigned 32-bit integer.

## Prototype

```
__SEGGER_RTL_U32 __udivmodsi4(__SEGGER_RTL_U32 num,  
                               __SEGGER_RTL_U32 den,  
                               __SEGGER_RTL_U32 *rem);
```

## Parameters

Parameter	Description
num	Dividend.
den	Divisor.
rem	Pointer to object that receives the remainder.

## Return value

Quotient.

[\\_\\_udivmoddi4\(\)](#)

## Description

Divide with remainder, unsigned 64-bit integer.

## Prototype

```
__SEGGER_RTL_U64 __udivmoddi4(__SEGGER_RTL_U64 num,  
                               __SEGGER_RTL_U64 den,  
                               __SEGGER_RTL_U64 *rem);
```

## Parameters

Parameter	Description
num	Dividend.
den	Divisor.
rem	Pointer to object that receives the remainder.

## Return value

Quotient.

[\\_\\_clzsi2\(\)](#)

## Description

Count leading zeros, 32-bit integer.

## Prototype

---

```
int __clzsi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument; x must not be zero.

Return value

Number of leading zeros in x.

[\\_\\_clzdi2\(\)](#)

Description

Count leading zeros, 64-bit integer.

Prototype

```
int __clzdi2(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Argument; x must not be zero.

Return value

Number of leading zeros in x.

[\\_\\_popcountsi2\(\)](#)

Description

Population count, 32-bit integer.

Prototype

```
int __popcountsi2(__SEGGER_RTL_U32 x);
```

Parameters

Parameter	Description
x	Argument.

Return value

Count of number of one bits in x.

---

### [\\_\\_popcountdi2\(\)](#)

#### Description

Population count, 64-bit integer.

#### Prototype

```
int __popcountdi2(__SEGGER_RTL_U64 x);
```

#### Parameters

Parameter	Description
x	Argument.

#### Return value

Count of number of one bits in x.

### [\\_\\_paritysi2\(\)](#)

#### Description

Parity, 32-bit integer.

#### Prototype

```
int __paritysi2(__SEGGER_RTL_U32 x);
```

#### Parameters

Parameter	Description
x	Argument.

#### Return value

- 1 number of one bits in x is odd.
- 0 number of one bits in x is even.

### [\\_\\_paritydi2\(\)](#)

#### Description

Parity, 64-bit integer.

#### Prototype

```
int __paritydi2(__SEGGER_RTL_U64 x);
```



## Parameters

Parameter	Description
x	Argument.

## Return value

- 1 number of one bits in x is odd.
- 0 number of one bits in x is even.

## Floating arithmetic

Function	Description
<code>__addsf3</code>	Add, float.
<code>__adddf3</code>	Add, double.
<code>__addtf3</code>	Add, long double.
<code>__subsf3</code>	Subtract, float.
<code>__subdf3</code>	Subtract, double.
<code>__subtf3</code>	Subtract, long double.
<code>__mulsf3</code>	Multiply, float.
<code>__muldf3</code>	Multiply, double.
<code>__multf3</code>	Multiply, long double.
<code>__divsf3</code>	Divide, float.
<code>__divdf3</code>	Divide, double.
<code>__divtf3</code>	Divide, long double.

### `__addsf3()`

## Description

Add, float.

## Prototype

```
float __addsf3(float x,  
               float y);
```

## Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

[\\_\\_adddf3\(\)](#)

Description

Add, double.

Prototype

```
double __adddf3(double x,  
                double y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

[\\_\\_addtf3\(\)](#)

Description

Add, long double.

Prototype

```
long double __addtf3(long double x,  
                    long double y);
```

Parameters

Parameter	Description
x	Augend.
y	Addend.

Return value

Sum.

### [\\_\\_subsf3\(\)](#)

Description

Subtract, float.

Prototype

```
float __subsf3(float x,  
               float y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

### [\\_\\_subdf3\(\)](#)

Description

Subtract, double.

Prototype

```
double __subdf3(double x,  
                double y);
```

Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

Return value

Difference.

### [\\_\\_subtf3\(\)](#)

Description

Subtract, long double.

## Prototype

```
long double __subtf3(long double x,  
                    long double y);
```

## Parameters

Parameter	Description
x	Minuend.
y	Subtrahend.

## Return value

Difference.

[\\_\\_mulsf3\(\)](#)

## Description

Multiply, float.

## Prototype

```
float __mulsf3(float x,  
              float y);
```

## Parameters

Parameter	Description
x	Multiplicand.
y	Multiplier.

## Return value

Product.

[\\_\\_muldf3\(\)](#)

## Description

Multiply, double.

## Prototype

```
double __muldf3(double x,  
               double y);
```

## Parameters

---

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

[\\_\\_multf3\(\)](#)

Description

Multiply, long double.

Prototype

```
long double __multf3(long double x,  
                    long double y);
```

Parameters

---

Parameter	Description
x	Multiplicand.
y	Multiplier.

Return value

Product.

[\\_\\_divsf3\(\)](#)

Description

Divide, float.

Prototype

```
float __divsf3(float x,  
              float y);
```

Parameters

---

Parameter	Description
x	Dividend.
y	Divisor.

Return value

---

Quotient.

[\\_\\_divdf3\(\)](#)

Description

Divide, double.

Prototype

```
double __divdf3(double x,  
                double y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

[\\_\\_divtf3\(\)](#)

Description

Divide, long double.

Prototype

```
long double __divtf3(long double x,  
                    long double y);
```

Parameters

Parameter	Description
x	Dividend.
y	Divisor.

Return value

Quotient.

### Floating conversions

Function	Description
<a href="#">__fixsfsi</a>	Convert float to int.

---

<code>__fixdfsi</code>	Convert double to int.
<code>__fixtfsi</code>	Convert long double to int.
<code>__fixsfdi</code>	Convert float to long long.
<code>__fixdfdi</code>	Convert double to long long.
<code>__fixtfdi</code>	Convert long double to long long.
<code>__fixunssfsi</code>	Convert float to unsigned.
<code>__fixunsdfsi</code>	Convert double to unsigned.
<code>__fixunstfsi</code>	Convert long double to int.
<code>__fixunssfdi</code>	Convert float to unsigned long long.
<code>__fixunsdfdi</code>	Convert double to unsigned long long.
<code>__fixunstfdi</code>	Convert long double to unsigned long long.
<code>__floatsisf</code>	Convert int to float.
<code>__floatsidf</code>	Convert int to double.
<code>__floatsitf</code>	Convert int to long double.
<code>__floatdisf</code>	Convert long long to float.
<code>__floatdidf</code>	Convert long long to double.
<code>__floatditf</code>	Convert long long to long double.
<code>__floatunsisf</code>	Convert unsigned to float.
<code>__floatunsidf</code>	Convert unsigned to double.
<code>__floatunsitf</code>	Convert unsigned to long double.
<code>__floatundisf</code>	Convert unsigned long long to float.
<code>__floatundidf</code>	Convert unsigned long long to double.
<code>__floatunditf</code>	Convert unsigned long long to long double.
<code>__extendsfdf2</code>	Extend float to double.
<code>__extendsftf2</code>	Extend float to long double.
<code>__extenddftf2</code>	Extend double to long double.
<code>__truncdfsf2</code>	Truncate double to float.
<code>__trunctfsf2</code>	Truncate long double to float.
<code>__trunctfdf2</code>	Truncate long double to double.

`__fixsfsi()`

Description

Convert float to int.

## Prototype

```
__SEGGER_RTL_I32 __fixsfsi(float x);
```

## Parameters

Parameter	Description
x	Floating value to convert.

## Return value

Integerized value.

[\\_\\_fixdfsi\(\)](#)

## Description

Convert double to int.

## Prototype

```
__SEGGER_RTL_I32 __fixdfsi(double x);
```

## Parameters

Parameter	Description
x	Floating value to convert.

## Return value

Integerized value.

[\\_\\_fixtfsi\(\)](#)

## Description

Convert long double to int.

## Prototype

```
__SEGGER_RTL_I32 __fixtfsi(long double x);
```

## Parameters

Parameter	Description
x	Floating value to convert.

## Return value



Integerized value.

[\\_\\_fixsfdi\(\)](#)

Description

Convert float to long long.

Prototype

```
__SEGGER_RTL_I64 __fixsfdi(float f);
```

Parameters

Parameter	Description
f	Floating value to convert.

Return value

Integerized value.

Notes

The RV32 compiler converts a float to a 64-bit integer by calling runtime support to handle it.

[\\_\\_fixdfdi\(\)](#)

Description

Convert double to long long.

Prototype

```
__SEGGER_RTL_I64 __fixdfdi(double x);
```

Parameters

Parameter	Description
x	Floating value to convert.

Return value

Integerized value.

Notes

RV32 always calls runtime for double to int64 conversion.

### [\\_\\_fixtfdi\(\)](#)

#### Description

Convert long double to long long.

#### Prototype

```
__SEGGER_RTL_I64 __fixtfdi(long double x);
```

#### Parameters

Parameter	Description
x	Floating value to convert.

#### Return value

Integerized value.

### [\\_\\_fixunssfsi\(\)](#)

#### Description

Convert float to unsigned.

#### Prototype

```
__SEGGER_RTL_U32 __fixunssfsi(float x);
```

#### Parameters

Parameter	Description
x	Float value to convert.

#### Return value

Integerized value.

### [\\_\\_fixunsdfs\(\)](#)

#### Description

Convert double to unsigned.

#### Prototype

```
__SEGGER_RTL_U32 __fixunsdfs(double x);
```

#### Parameters

---

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

[`\_\_fixunstfsi\(\)`](#)

Description

Convert long double to int.

Prototype

```
int __fixunstfsi(long double x);
```

Parameters

---

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

[`\_\_fixunssfdi\(\)`](#)

Description

Convert float to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunssfdi(float f);
```

Parameters

---

Parameter	Description
f	Float value to convert.

Return value

Integerized value.

[`\_\_fixunsdfdi\(\)`](#)

Description

---

Convert double to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunsdftdi(double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

[\\_\\_fixunstfdi\(\)](#)

Description

Convert long double to unsigned long long.

Prototype

```
__SEGGER_RTL_U64 __fixunstfdi(long double x);
```

Parameters

Parameter	Description
x	Float value to convert.

Return value

Integerized value.

[\\_\\_floatsisf\(\)](#)

Description

Convert int to float.

Prototype

```
float __floatsisf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

[\\_\\_floatsidf\(\)](#)

Description

Convert int to double.

Prototype

```
double __floatsidf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

[\\_\\_floatsitf\(\)](#)

Description

Convert int to long double.

Prototype

```
long double __floatsitf(__SEGGER_RTL_I32 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

[\\_\\_floatdisf\(\)](#)

Description

Convert long long to float.

Prototype

---

```
float __floatdisf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

[\\_\\_floatdidf\(\)](#)

Description

Convert long long to double.

Prototype

```
double __floatdidf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

[\\_\\_floatditf\(\)](#)

Description

Convert long long to long double.

Prototype

```
long double __floatditf(__SEGGER_RTL_I64 x);
```

Parameters

Parameter	Description
x	Integer value to convert.

Return value

Floating value.

---

### [\\_\\_floatunsisf\(\)](#)

#### Description

Convert unsigned to float.

#### Prototype

```
float __floatunsisf(__SEGGER_RTL_U32 x);
```

#### Parameters

Parameter	Description
x	Integer value to convert.

#### Return value

Floating value.

### [\\_\\_floatunsidf\(\)](#)

#### Description

Convert unsigned to double.

#### Prototype

```
double __floatunsidf(__SEGGER_RTL_U32 x);
```

#### Parameters

Parameter	Description
x	Unsigned value to convert.

#### Return value

Double value.

### [\\_\\_floatunsitf\(\)](#)

#### Description

Convert unsigned to long double.

#### Prototype

```
long double __floatunsitf(__SEGGER_RTL_U32 x);
```

#### Parameters

---

Parameter	Description
x	Unsigned value to convert.

Return value

Long double value.

[`\_\_floatundisf\(\)`](#)

Description

Convert unsigned long long to float.

Prototype

```
float __floatundisf(__SEGGER_RTL_U64 x);
```

Parameters

---

Parameter	Description
x	Unsigned long long value to convert.

Return value

Float value.

[`\_\_floatundidf\(\)`](#)

Description

Convert unsigned long long to double.

Prototype

```
double __floatundidf(__SEGGER_RTL_U64 x);
```

Parameters

---

Parameter	Description
x	Unsigned long long value to convert.

Return value

Double value.

[`\_\_floatunditf\(\)`](#)

Description

---



Convert unsigned long long to long double.

Prototype

```
long double __floatunditf(__SEGGER_RTL_U64 x);
```

Parameters

Parameter	Description
x	Unsigned long long value to convert.

Return value

Long double value.

[\\_\\_extendsfdf2\(\)](#)

Description

Extend float to double.

Prototype

```
double __extendsfdf2(float x);
```

Parameters

Parameter	Description
x	Float value to extend.

Return value

Double value.

[\\_\\_extendsftf2\(\)](#)

Description

Extend float to long double.

Prototype

```
long double __extendsftf2(float x);
```

Parameters

Parameter	Description
x	Float value to extend.

Return value

Double value.

[\\_\\_extenddftf2\(\)](#)

Description

Extend double to long double.

Prototype

```
long double __extenddftf2(double x);
```

Parameters

Parameter	Description
x	Double value to extend.

Return value

Long double value.

[\\_\\_truncdfsf2\(\)](#)

Description

Truncate double to float.

Prototype

```
float __truncdfsf2(double x);
```

Parameters

Parameter	Description
x	Double value to truncate.

Return value

Float value.

[\\_\\_trunctfdf2\(\)](#)

Description

Truncate long double to double.

Prototype

```
double __trunctfdf2(long double x);
```

Parameters

Parameter	Description
x	Long double value to truncate.

Return value

Double value.

[\\_\\_trunctfsf2\(\)](#)

Description

Truncate long double to float.

Prototype

```
float __trunctfsf2(long double x);
```

Parameters

Parameter	Description
x	Long double value to truncate.

Return value

Float value.

### Floating comparisons

Function	Description
<a href="#">__eqsf2</a>	Equal, float.
<a href="#">__eqdf2</a>	Equal, double.
<a href="#">__eqtf2</a>	Equal, long double.
<a href="#">__nesf2</a>	Not equal, float.
<a href="#">__nedf2</a>	Not equal, double.
<a href="#">__netf2</a>	Not equal, long double.
<a href="#">__ltsf2</a>	Less than, float.
<a href="#">__ltdf2</a>	Less than, double.
<a href="#">__ltdf2</a>	Less than, long double.
<a href="#">__lesf2</a>	Less than or equal, float.
<a href="#">__ledf2</a>	Less than or equal, double.

`__letf2`      Less than or equal, long double.  
`__gtsf2`      Greater than, float.  
`__gtdf2`      Greater than, double.  
`__gttf2`      Greater than, long double.  
`__gesf2`      Greater than or equal, float.  
`__gedf2`      Greater than or equal, double.  
`__getf2`      Greater than or equal, long double.

### `__eqsf2()`

Description

Equal, float.

Prototype

```
int __eqsf2(float x,  
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

### `__eqdf2()`

Description

Equal, double.

Prototype

```
int __eqdf2(double x,  
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.

y            Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

[\\_\\_eqtf2\(\)](#)

Description

Equal, long double.

Prototype

```
int __eqtf2(long double x,  
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

[\\_\\_nesf2\(\)](#)

Description

Not equal, float.

Prototype

```
int __nesf2(float x,  
            float y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

## [\\_\\_nedf2\(\)](#)

### Description

Not equal, double.

### Prototype

```
int __nedf2(double x,  
            double y);
```

### Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

### Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

## [\\_\\_netf2\(\)](#)

### Description

Not equal, long double.

### Prototype

```
int __netf2(long double x,  
            long double y);
```

### Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

### Return value

Return = 0 if both operands are non-NaN and a = b (GNU three-way boolean).

## [\\_\\_ltsf2\(\)](#)

### Description

Less than, float.

## Prototype

```
int __ltsf2(float x,  
           float y);
```

## Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

## Return value

Return < 0 if both operands are non-NaN and  $a < b$  (GNU three-way boolean).

[\\_\\_ltdf2\(\)](#)

## Description

Less than, double.

## Prototype

```
int __ltdf2(double x,  
           double y);
```

## Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

## Return value

Return < 0 if both operands are non-NaN and  $a < b$  (GNU three-way boolean).

[\\_\\_ltdf2\(\)](#)

## Description

Less than, long double.

## Prototype

```
int __ltdf2(long double x,  
           long double y);
```

## Parameters

---

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return < 0 if both operands are non-NaN and  $a < b$  (GNU three-way boolean).

[\\_\\_lesf2\(\)](#)

Description

Less than or equal, float.

Prototype

```
int __lesf2(float x,
            float y);
```

Parameters

---

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return  $\leq 0$  if both operands are non-NaN and  $a < b$  (GNU three-way boolean).

[\\_\\_ledf2\(\)](#)

Description

Less than or equal, double.

Prototype

```
int __ledf2(double x,
            double y);
```

Parameters

---

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

---



Return  $\leq 0$  if both operands are non-NaN and  $a < b$  (GNU three-way boolean).

### [\\_\\_letf2\(\)](#)

#### Description

Less than or equal, long double.

#### Prototype

```
int __letf2(long double x,  
            long double y);
```

#### Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

#### Return value

Return  $\leq 0$  if both operands are non-NaN and  $a < b$  (GNU three-way boolean).

### [\\_\\_gtsf2\(\)](#)

#### Description

Greater than, float.

#### Prototype

```
int __gtsf2(float x,  
            float y);
```

#### Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

#### Return value

Return  $> 0$  if both operands are non-NaN and  $a > b$  (GNU three-way boolean).

### [\\_\\_gtdf2\(\)](#)

#### Description

Greater than, double.

Prototype

```
int __gtdf2(double x,  
            double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and  $a > b$  (GNU three-way boolean).

[\\_\\_gttf2\(\)](#)

Description

Greater than, long double.

Prototype

```
int __gttf2(long double x,  
            long double y);
```

Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return > 0 if both operands are non-NaN and  $a > b$  (GNU three-way boolean).

[\\_\\_gesf2\(\)](#)

Description

Greater than or equal, float.

Prototype

```
int __gesf2(float x,  
            float y);
```

## Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

## Return value

Return  $\geq 0$  if both operands are non-NaN and  $a \geq b$  (GNU three-way boolean).

## [\\_\\_gedf2\(\)](#)

## Description

Greater than or equal, double.

## Prototype

```
int __gedf2(double x,  
            double y);
```

## Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

## Return value

Return  $\geq 0$  if both operands are non-NaN and  $a \geq b$  (GNU three-way boolean).

## [\\_\\_getf2\(\)](#)

## Description

Greater than or equal, long double.

## Prototype

```
int __getf2(long double x,  
            long double y);
```

## Parameters

Parameter	Description
x	Left-hand operand.
y	Right-hand operand.

Return value

Return  $\geq 0$  if both operands are non-NaN and  $a \geq b$  (GNU three-way boolean).

## External function interface

This section summarises the functions to be provided by the implementor when integrating Nuclei C Runtime Library into an application or library.

### I/O functions

Function	Description
<code>__SEGGER_RTL_X_file_read</code>	Read data from file.
<code>__SEGGER_RTL_X_file_write</code>	Write data to file.
<code>__SEGGER_RTL_X_file_unget</code>	Push character back to file.

#### `__SEGGER_RTL_X_file_read()`

Description

Read data from file.

Prototype

```
int __SEGGER_RTL_X_file_read(  
    char * s,  
    unsigned len);
```

Parameters

Parameter	Description
<code>stream</code>	Pointer to file to read from.
<code>s</code>	Pointer to object that receives the input.
<code>len</code>	Number of characters to read from file.

Return value

$\geq 0$  Success.

$< 0$  Failure.

Additional information

This reads `len` octets from the file stream into the object pointed to by `s`.

## `__SEGGER_RTL_X_file_write()`

### Description

Write data to file.

### Prototype

```
int __SEGGER_RTL_X_file_write(
    const char * s,
    unsigned len);
    __SEGGER_RTL_FILE *stream,
```

### Parameters

Parameter	Description
stream	Pointer to stream to write to.
s	Pointer to object to write to stream.
len	Number of characters to write to the stream.

### Return value

$\geq 0$  Success.  
 $< 0$  Failure.

### Additional information

This writes len octets to the file stream from the object pointed to by s.

## `__SEGGER_RTL_X_file_unget()`

### Description

Push character back to file.

### Prototype

```
int __SEGGER_RTL_X_file_unget(
    __SEGGER_RTL_FILE *stream,
    int c);
```

### Parameters

Parameter	Description
stream	File to push character to.
c	Character to push back to file.

### Return value

= EOF Failed to push character back.  
≠ EOF The character pushed back to the file.

#### Additional information

This function pushes the character *c* back to the file so that it can be read again. If *c* is EOF, the function fails and EOF is returned. One character of pushback is guaranteed; if more than one character is pushed back without an intervening read, the pushback may fail.

### Heap protection functions

Function	Description
<code>__SEGGER_RTL_X_heap_lock</code>	Lock heap.
<code>__SEGGER_RTL_X_heap_unlock</code>	Unlock heap.

#### `__SEGGER_RTL_X_heap_lock()`

##### Description

Lock heap.

##### Prototype

```
void __SEGGER_RTL_X_heap_lock(void);
```

#### Additional information

This function is called to lock access to the heap before allocation or deallocation is processed. This is only required for multitasking systems where heap operations may possibly be called from different threads.

#### `__SEGGER_RTL_X_heap_unlock()`

##### Description

Unlock heap.

##### Prototype

```
void __SEGGER_RTL_X_heap_unlock(void);
```

#### Additional information

This function is called to unlock access to the heap after allocation or deallocation has completed. This is only required for multitasking systems where heap operations may possibly be called from different threads.

## Error and assertion functions

Function	Description
<code>__SEGGER_RTL_X_assert</code>	User-defined behavior for the assert macro.
<code>__SEGGER_RTL_X_errno_addr</code>	Return pointer to object holding errno.

### `__SEGGER_RTL_X_assert()`

#### Description

User-defined behavior for the assert macro.

#### Prototype

```
void __SEGGER_RTL_X_assert(const char * expr,
                           const char * filename,
                           int      line);
```

#### Parameters

Parameter	Description
<code>expr</code>	Stringized expression that caused failure.
<code>filename</code>	Filename of the source file where the failure was signaled.
<code>line</code>	Line number of the failed assertion.

#### Additional information

The default implementation of `__SEGGER_RTL_X_assert()` prints the filename, line, and error message to standard output and then calls `abort()`.

`__SEGGER_RTL_X_assert()` is defined as a weak function and can be replaced by user code.

### `__SEGGER_RTL_X_errno_addr()`

#### Description

Return pointer to object holding errno.

#### Prototype

```
int *__SEGGER_RTL_X_errno_addr(void);
```

#### Return value

Pointer to errno object.

#### Additional information

The default implementation of this function is to return the address of a variable declared with the `__SEGGER_RTL_THREAD` storage class. Thus, for multithreaded environments that implement thread-local variables through `__SEGGER_RTL_THREAD`, each thread receives its own thread-local `errno`.

It is beyond the scope of this manual to describe how thread-local variables are implemented by the compiler and any associated real-time operating system.

When `__SEGGER_RTL_THREAD` is defined as an empty macro, this function returns the address of a singleton `errno` object.

## RTC functions

Function	Description
<code>__SEGGER_RTL_X_set_time_of_day</code>	Set RTC time.
<code>__SEGGER_RTL_X_get_time_of_day</code>	Get RTC time.

### `__SEGGER_RTL_X_set_time_of_day()`

Description

Set RTC time.

Prototype

```
int __SEGGER_RTL_X_set_time_of_day(const struct timeval *__tp);
```

Parameters

`tp` - Pointer to `timeval`.

Return value

return 0 for success, or -1 for failure.

### `__SEGGER_RTL_X_get_time_of_day()`

Description

Get RTC time.

Prototype

```
int __SEGGER_RTL_X_get_time_of_day(struct timeval *__tp);
```



## Parameters

tp - Pointer to timeval.

## Return value

return 0 for success, or -1 for failure.

## Locale functions

Function	Description
<code>__SEGGER_RTL_X_find_locale</code>	Find locale.

### `__SEGGER_RTL_X_find_locale()`\*

## Description

Find locale.

## Prototype

```
const __SEGGER_RTL_locale_t * __SEGGER_RTL_X_find_locale(const char *locale);
```

## Parameters

locale - Pointer to zero-terminated locale name.

## Return value

Returns a pointer to a locale or NULL if none can be found.