IAR C Library Functions

Reference Guide

IARCLIB-1

IAR
SYSTEMS

COPYRIGHT NOTICE

DISCLAIMER

The information in this document is subject to change without notice and does not represent a commitment on any part of IAR Systems. While the information contained herein is assumed to be accurate, IAR Systems assumes no responsibility for any errors or omissions.

In no event shall IAR Systems, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

TRADEMARKS

IAR and C-SPY are registered trademarks of IAR Systems. IAR Embedded Workbench, IAR XLINK Linker, and IAR XLIB Librarian are trademarks of IAR Systems. Microsoft is a registered trademark, and Windows is a trademark of Microsoft Corporation.

All other product names are trademarks or registered trademarks of their respective owners.

EDITION NOTICE

First edition: September 2000

Part number: IARCLIB-1

C library functions reference

This guide gives an alphabetical list of the C library functions, including a full description of their operation and options available for each one.

Descriptions of C library functions

Each function description contains the following information:

- Function name
  The name of the C library function.
- Declaration
  The C library declaration.
- Parameters
  Details of each parameter in the declaration.
- Return value
  The value, if any, returned by the function.
- Description
  A detailed description covering the function's most general use. This includes information about what the function is useful for, and a discussion of any special conditions and common pitfalls.
- Header filename
  The function header filename.
- Examples
  One or more examples illustrating how the function can be used.

The following sections contain full reference information for each C library function.

abort `void abort(void)`

Parameters

None.

Return value

None.

Description

Terminates the program abnormally and does not return to the caller. This function calls the funciton `exit`, and by default the entry for this resides in `CSTARTUP`.

Header file

`stdlib.h`

`abs` `int abs(int j)`

Parameters

`j`	An `int` value.

Return value

An `int` having the absolute value of *`j`*.

Description

Computes the absolute value of *`j`*.

Header file

`stdlib.h`

`acos` `double acos(double arg)`

Parameters

`arg`	A `double` in the range [`-1`,`+1`].

Return value

The `double` arc cosine of *`arg`*, in the range [`0,pi`].

Description

Computes the principal value in radians of the arc cosine of *`arg`*.

Header file

`math.h`

asin double asin(double *arg*)

Parameters

arg	A double in the range [-1,+1].

Return value

The double arc sine of *arg*, in the range [-pi/2,+pi/2].

Description

Computes the principal value in radians of the arc sine of *arg*.

Header file

math.h

assert void assert (int *expression*)

Parameters

expression	An expression to be checked.

Return value

None.

Description

This is a macro that checks an expression. If it is false it prints a message to stderr and calls abort.

The message has the following format:

File *name*; line *num* # Assertion failure "*expression*"

To ignore assert calls put a #define NDEBUG statement before the #include <assert.h> statement.

Header file

assert.h

atan

`double atan(double arg)`

Parameters

arg	A `double` value.

Return value

The `double` arc tangent of *arg*, in the range [`-pi/2`,`pi/2`].

Description

Computes the arc tangent of *arg*.

Header file

`math.h`

atan2

`double atan2(double arg1, double arg2)`

Parameters

arg1	A `double` value.
arg2	A `double` value.

Return value

The double arc tangent of *arg1*/*arg2*, in the range [`-pi`,`pi`].

Description

Computes the arc tangent of *arg1*/*arg2*, using the signs of both arguments to determine the quadrant of the return value.

Header file

`math.h`

atof

`double atof(const char *nptr)`

Parameters

nptr	A pointer to a string containing a number in ASCII form.

Return value

The `double` number found in the string.

Description

Converts the string pointed to by *`nptr`* to a double-precision floating-point number, skipping white space and terminating upon reaching any unrecognized character.

Header file

`stdlib.h`

Examples

`"   -3K"` gives `-3.00`

`".0006"` gives `0.0006`

`"1e-4"` gives `0.0001`

`atoi` `int atoi(const char *nptr)`

Parameters

`nptr`	A pointer to a string containing a number in ASCII form.

Return value

The `int` number found in the string.

Description

Converts the ASCII string pointed to by *`nptr`* to an integer, skipping white space and terminating upon reaching any unrecognized character.

Header file

`stdlib.h`

Examples

`"   -3K"` gives `-3`

`"6"` gives `6`

`"149"` gives `149`

atol

```
long atol(const char *nptr)
```

Parameters

nptr	A pointer to a string containing a number in ASCII form.

Return value

The `long` number found in the string.

Description

Converts the number found in the ASCII string pointed to by *nptr* to a long integer value, skipping white space and terminating upon reaching any unrecognized character.

Header file

`stdlib.h`

Examples

`"   -3K"` gives `-3`

`"6"` gives `6`

`"149"` gives `149`

bsearch

```
void *bsearch(const void *key, const void *base, size_t
       nmemb, size_t size, int (*compare) (const void *_key,
       const void *_base));
```

Parameters

key	Pointer to the searched for object.
base	Pointer to the array to search.
nmemb	Dimension of the array pointed to by *base*.
size	Size of the array elements.
compare	The comparison function which takes two arguments and returns: <0 (negative value) if *_key* is less than *_base* 0 if *_key* equals *_base* >0 (positive value) if *_key* is greater than *_base*

Return value

Result	Value
Successful	A pointer to the element of the array that matches the key.
Unsuccessful	Null.

Table 1: bsearch return value

Description

Searches an array of *nmemb* objects, pointed to by *base*, for an element that matches the object pointed to by *key*.

Header file

stdlib.h

calloc

`void *calloc(size_t nelem, size_t elsize)`

Parameters

nelem	The number of objects.
elsize	A value of type *size_t* specifying the size of each object.

Return value

Result	Value
Successful	A pointer to the start (lowest address) of the memory block.
Unsuccessful	Zero if there is no memory block of the required size or greater available.

Table 2: calloc return values

Description

Allocates a memory block for an array of objects of the given size. To ensure portability, the size is not given in absolute units of memory such as bytes, but in terms of a size or sizes returned by the `sizeof` function.

The availability of memory depends on the default heap size, see the *IAR C Compiler Reference Guide*.

Header file

stdlib.h

ceil `double ceil(double *arg*)`

Parameters

arg	A `double` value.

Return value

A `double` having the smallest integral value greater than or equal to *arg*.

Description

Computes the smallest integral value greater than or equal to *arg*.

Header file

`math.h`

cos `double cos(double *arg*)`

Parameters

arg	A `double` value in radians.

Return value

The `double` cosine of *arg*.

Description

Computes the cosine of *arg* radians.

Header file

`math.h`

cosh `double cosh(double *arg*)`

Parameters

arg	A `double` value in radians.

Return value

The `double` hyperbolic cosine of *arg*.

Description

Computes the hyperbolic cosine of *arg* radians.

Header file

math.h

---

div div_t div(int *numer*, int *denom*)

Parameters

numer	The int numerator.
demon	The int denominator.

Return value

A structure of type div_t holding the quotient and remainder results of the division.

Description

Divides the numerator *numer* by the denominator *denom*. The type div_t is defined in stdlib.h.

If the division is inexact, the quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. The results are defined such that:

```
quot * denom + rem == numer
```

Header file

math.h

---

exit void exit(int *status*)

Parameters

status	An int status value.

Return value

None.

Description

Terminates the program normally. This function does not return to the caller. This function entry resides by default in `CSTARTUP`.

Header file

`stdlib.h`

`exp` `double exp(double *arg*)`

Parameters

`arg`	A `double` value.

Return value

A `double` with the value of the exponential function of *`arg`*.

Description

Computes the exponential function of *`arg`*.

Header file

`math.h`

`exp10` `double exp10(double *arg*)`

Parameters

`arg`	A `double` value.

Return value

A `double` with the value of 10^*`arg`*.

Description

Computes the value of 10^*`arg`*.

Header file

`iccext.h`

fabs `double fabs(double` *`arg`*`)`

Parameters

arg	A `double` value.

Return value

The `double` absolute value of *arg*.

Description

Computes the absolute value of the floating-point number *arg*.

Header file

`math.h`

floor `double floor(double` *`arg`*`)`

Parameters

arg	A `double` value.

Return value

A `double` with the value of the largest integer less than or equal to *arg*.

Description

Computes the largest integral value less than or equal to *arg*.

Header file

`math.h`

fmod `double fmod(double` *`arg1`*`, double` *`arg2`*`)`

Parameters

arg1	The `double` numerator.
arg2	The `double` denominator.

Return value

The `double` remainder of the division *`arg1/arg2`*.

Description

Computes the remainder of *`arg1/arg2`*, i.e. the value *`arg1`*–i*****`arg2`, for some integer `i` such that, if *`arg2`* is non-zero, the result has the same sign as *`arg1`* and magnitude less than the magnitude of *`arg2`*.

Header file

`math.h`

`free` `void free(void *ptr)`

Parameters

`ptr`	A pointer to a memory block previously allocated by `malloc`, `calloc`, or `realloc`.

Return value

None.

Description

Frees the memory used by the object pointed to by *`ptr. ptr`* must earlier have been assigned a value from `malloc`, `calloc`, or `realloc`.

Header file

`stdlib.h`

`frexp` `double frexp(double arg1, int *arg2)`

Parameters

`arg1`	Floating-point number to be split.
`arg2`	Pointer to an integer to contain the exponent of *`arg1`*.

Return value

The `double` mantissa of *`arg1`*, in the range `0.5` to `1.0`.

Description

Splits the floating-point number *arg1* into an exponent stored in **arg2*, and a mantissa which is returned as the value of the function.

The values are as follows:

mantissa * 2^{exponent} = value

Header file

`math.h`

---

getchar `int getchar(void)`

Parameters

None.

Return value

An `int` with the ASCII value of the next character from the standard input stream.

Description

Gets the next character from the standard input stream.

You should customize this function for the particular target hardware configuration. The function is supplied in source format in the file `getchar.c`.

Header file

`stdio.h`

---

gets `char *gets(char *s)`

Parameters

s	A pointer to the string that is to receive the input.

Return value

Result	Value
Successful	A pointer equal to *s*.
Unsuccessful	Null.

Table 3: gets return values

Description

Gets the next string from standard input and places it in the string pointed to. The string is terminated by end-of-line or end-of-file. The end-of-line character is replaced by zero.

This function calls `getchar`, which must be adapted for the particular target hardware configuration.

Header file

`stdio.h`

`isalnum` `int isalnum(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` that is non-zero if *c* is a letter or digit, else zero.

Description

Tests whether a character is a letter or digit.

Header file

`ctype.h`

`isalpha` `int isalpha(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is letter, else zero.

Description

Tests whether a character is a letter.

Header file

`ctype.h`

`iscntrl` `int iscntrl(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is a control code, else zero.

Description

Tests whether a character is a control character.

Header file

`ctype.h`

`isdigit` `int isdigit(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is a digit, else zero.

Description

Tests whether a character is a decimal digit.

Header file

`ctype.h`

`isgraph` `int isgraph(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is a printable character other than space, else zero.

Description

Tests whether a character is a printable character other than space.

Header file

`ctype.h`

`islower` `int islower(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is lowercase, else zero.

Description

Tests whether a character is a lowercase letter.

Header file

`ctype.h`

`isprint` `int isprint(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is a printable character, including space, else zero.

Description

Tests whether a character is a printable character, including space.

Header file

`ctype.h`

`ispunct` `int ispunct(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` that is non-zero if *c* is printable character other than space, digit, or letter, else zero.

Description

Tests whether a character is a printable character other than space, digit, or letter.

Header file

`ctype.h`

`isspace` `int isspace (int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is a white-space character, else zero.

Description

Tests whether a character is a white-space character, that is, one of the following:

Character	Symbol
Space	`' '`
Formfeed	`\f`

Table 4: isspace

Character	Symbol
Newline	`\n`
Carriage return	`\r`
Horizontal tab	`\t`
Vertical tab	`\v`

Table 4: isspace

Header file

`ctype.h`

`isupper` `int isupper(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is uppercase, else zero.

Description

Tests whether a character is an uppercase letter.

Header file

`ctype.h`

`isxdigit` `int isxdigit(int c)`

Parameters

c	An `int` representing a character.

Return value

An `int` which is non-zero if *c* is a digit in uppercase or lowercase, else zero.

Description

Tests whether the character is a hexadecimal digit in uppercase or lowercase, that is, one of `0-9`, `a-f`, or `A-F`.

Header file

`ctype.h`

---

labs `long int labs(long int j)`

Parameters

j	A `long int` value.

Return value

The `long int` absolute value of *j*.

Description

Computes the absolute value of the long integer *j*.

Header file

`stdlib.h`

---

ldexp `double ldexp(double arg1,int arg2)`

Parameters

arg1	The `double` multiplier value.
arg2	The `int` power value.

Return value

The `double` value of *arg1* multiplied by two raised to the power of *arg2*.

Description

Computes the value of the floating-point number multiplied by 2 raised to a power.

Header file

`math.h`

ldiv ldiv_t ldiv(long int *numer*, long int *denom*)

Parameters

numer	The long int numerator.
denom	The long int denominator.

Return value

A struct of type ldiv_t holding the quotient and remainder of the division.

Description

Divides the numerator *numer* by the denominator *denom*. The type ldiv_t is defined in stdlib.h.

If the division is inexact, the quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. The results are defined such that:

```
quot * denom + rem == numer
```

Header file

stdlib.h

log double log(double *arg*)

Parameters

arg	A double value.

Return value

The double natural logarithm of *arg*.

Description

Computes the natural logarithm of a number.

Header file

math.h

`log10` `double log10(double` *`arg`*`)`

Parameters

`arg`	A `double` number.

Return value

The `double` base-10 logarithm of *`arg`*.

Description

Computes the base-10 logarithm of a number.

Header file

`math.h`

`longjmp` `void longjmp(jmp_buf` *`env`*`, int` *`val`*`)`

Parameters

`env`	A `struct` of type `jmp_buf` holding the environment set by `setjmp`.
`val`	The `int` value to be returned by the corresponding `setjmp`.

Return value

None.

Description

Restores the environment previously saved by `setjmp`. This causes program execution to continue as a return from the corresponding `setjmp`, returning the value *`val`*.

Header file

`setjmp.h`

malloc `void *malloc(size_t size)`

Parameters

size	A `size_t` object specifying the size of the object.

Return value

Result	Value
Successful	A pointer to the start (lowest byte address) of the memory block.
Unsuccessful	Zero, if there is no memory block of the required size or greater available.

Table 5: malloc return values

Description

Allocates a memory block for an object of the specified size.

The availability of memory depends on the size of the heap. For more information about changing the heap size, see the *IAR C Compiler Reference Guide*.

Header file

`stdlib.h`

memchr `void *memchr(const void *s, int c, size_t n)`

Parameters

s	A pointer to an object.
c	An `int` representing a character.
n	A value of type `size_t` specifying the size of each object.

Return value

Result	Value
Successful	A pointer to the first occurrence of *c* in the *n* characters pointed to by *s*.
Unsuccessful	Null.

Table 6: memchr return values

Description

Searches for the first occurrence of a character in a pointed-to region of memory of a given size.

Both the single character and the characters in the object are treated as unsigned.

Header file

`string.h`

---

memcmp `int memcmp(const void *s1, const void *s2, size_t n`

Parameters

s1	A pointer to the first object.
s2	A pointer to the second object.
n	A value of type `size_t` specifying the size of each object.

Return value

An integer indicating the result of comparison of the first *n* characters of the object pointed to by *s1* with the first *n* characters of the object pointed to by *s2*:

Return value	Meaning
>0	s1 > s2
=0	s1 = s2
<0	s1 < s2

Table 7: memcmp return values

Description

Compares the first *n* characters of two objects.

Header file

`string.h`

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

Parameters

s1	A pointer to the destination object.
s2	A pointer to the source object.
n	The number of characters to be copied.

Return value

s1.

Description

Copies a specified number of characters from a source object to a destination object.

If the objects overlap, the result is undefined, so `memmove` should be used instead.

Header file

`string.h`

memmove `void *memmove(void *s1, const void *s2, size_t n)`

Parameters

s1	A pointer to the destination object.
s2	A pointer to the source object.
n	The number of characters to be copied.

Return value

s1.

Description

Copies a specified number of characters from a source object to a destination object.

Copying takes place as if the source characters are first copied into a temporary array that does not overlap either object, and then the characters from the temporary array are copied into the destination object.

Header file

`string.h`

---

memset `void *memset(void *s, int c, size_t n)`

Parameters

s	A pointer to the destination object.
c	An `int` representing a character.
n	The size of the object.

Return value

s.

Description

Copies a character (converted to an `unsigned char`) into each of the first specified number of characters of the destination object.

Header file

`string.h`

---

modf `double modf(double value, double *iptr)`

Parameters

value	A `double` value.
iptr	A pointer to the `double` that is to receive the integral part of value.

Return value

The fractional part of *value*.

Description

Computes the fractional and integer parts of *value*. The sign of both parts is the same as the sign of *value*.

Header file

`math.h`

---

pow `double pow(double arg1, double arg2)`

Parameters

arg1	The `double` number.
arg2	The `double` power.

Return value

arg1 raised to the power of *arg2*.

Description

Computes a number raised to a power.

Header file

`math.h`

---

printf `int printf(const char *format, …)`

Parameters

format	A pointer to the format string.
…	The optional values that are to be printed under the control of *format*.

Return value

Result	Value
Successful	The number of characters written.
Unsuccessful	A negative value, if an error occurred.

Table 8: printf return values

Description

Writes formatted data to the standard output stream, returning the number of characters written, or a negative value if an error occurred.

Since a complete formatter demands a lot of space there are several different formatters to choose between. For more information, see the see the *IAR C Compiler Reference Guide*.

The parameter *format* is a string consisting of a sequence of characters to be printed and conversion specifications. Each conversion specification causes the next successive argument following the *format* string to be evaluated, converted, and written.

The form of a conversion specification is as follows:

```
% [flags] [field_width] [.precision] [length_modifier]
conversion
```

Items inside [ ] are optional.

Flags

The *flags* are as follows:

Flag	Effect	
-	Left adjusted field.	
+	Signed values will always begin with plus or minus sign.	
space	Values will always begin with minus or space.	
#	Alternatives:	
	octal	First digit will always be a zero.
	G g	Decimal point printed and trailing zeros kept.
	E e f	Decimal point printed.
	X	Non-zero values prefixed with 0X.
X	Non-zero values prefixed with 0X.	
0	Zero padding to field width (for d, i, o, u, x, X, e, E, f, g, and G specifiers).	

Table 9: printf flags

Field width

The *field_width* is the number of characters to be printed in the field. The field will be padded with space if needed. A negative value indicates a left-adjusted field. A field width of * stands for the value of the next successive argument, which should be an integer.

Precision

The *precision* is the number of digits to print for integers (d, i, o, u, x, and X), the number of decimals printed for floating-point values (e, E, and f), and the number of significant digits for g and G conversions. A field width of * stands for the value of the next successive argument, which should be an integer.

Length modifier

The effect of each *length_modifier* is as follows:

Modifier	Use
h	Before d, i, u, x, X, or o specifiers to denote a short int or unsigned short int value.
l	Before d, i, u, x, X, or o specifiers to denote a long integer or unsigned long value.
L	Before e, E, f, g, or G specifiers to denote a long double value.

Table 10: printf length modifiers

Conversion

The result of each value of *conversion* is as follows:

Conversion	Result
d	Signed decimal value.
i	Signed decimal value.
o	Unsigned octal value.
u	Unsigned decimal value.
x	Unsigned hexadecimal value, using lower case (0–9, a–f).
X	Unsigned hexadecimal value, using upper case (0–9, A–F).
e	Double value in the style [-]d.ddde+dd.
E	Double value in the style [-]d.dddE+dd.
f	Double value in the style [-]ddd.ddd.
g	Double value in the style of f or e, whichever is the more appropriate.
G	Double value in the style of F or E, whichever is the more appropriate.
C	Single character constant.
s	String constant.
p	Pointer value (address).

Table 11: printf conversion

Conversion	Result
n	No output, but stores the number of characters written so far in the integer pointed to by the next argument.
%	% character.

Table 11: printf conversion

Note: Promotion rules convert all `char` and `short int` arguments to `int` while `floats` are converted to `double`.

`printf` calls the library function `putchar`, which must be adapted for the target hardware configuration.

The source of `printf` is provided in the file `printf.c`. The source of a reduced version that uses less program space and stack is provided in the file `intwri.c`.

Header file

```
stdio.h
```

Examples

After the following C statements:

```
int i=6, j=-6;
char *p = "ABC";
long l=100000;
float f1 = 0.0000001;
f2 = 750000;
double d = 2.2;
```

the effect of different `printf` function calls is shown in the following table where ° represents space:

Statement	Output	Characters output
printf("%c",p[1])	B	1
printf("%d",i)	6	1
printf("%3d",i)	°6	3
printf("%.3d",i)	°°6	3
printf("%-10.3d",i)	006°°°°°°°	10
printf("%10.3d",i)	°°°°°°°006	10
printf("Value=%+3d",i)	Value=°+6	9
printf("%10.*d",i,j)	°°°-000006	10
printf("String=[%s]",p)	String=[ABC]	12

Table 12: printf function calls

Statement	Output	Characters output
printf("Value=%lX",l)	Value=186A0	11
printf("%f",f1)	0.000000	8
printf("%f",f2)	750000.000000	13
printf("%e",f1)	1.000000e-07	12
printf("%16e",d)	°°°°2.200000e+00	16
printf("%.4e",d)	2.2000e+00	10
printf("%g",f1)	1e-07	5
printf("%g",f2)	750000	6
printf("%g",d)	2.2	3

Table 12: printf function calls

putchar

```
int putchar(int value)
```

Parameters

value	The int representing the character to be put.

Return value

Result	Value
Successful	*value*.
Unsuccessful	The EOF macro.

Table 13: putchar return values

Description

Writes a character to standard output.

You should customize this function for the particular target hardware configuration. The function is supplied in source format in the file putchar.c.

This function is called by printf.

Header file

stdio.h

```
puts  int puts(const char *s)
```

Parameters

s	A pointer to the string to be put.

Return value

Result	Value
Successful	A non-negative value.
Unsuccessful	-1 if an error occurred.

Table 14: puts return values

Description

Writes a string followed by a new-line character to the standard output stream.

Header file

`stdio.h`

```
qsort  void qsort (const void *base, size_t nmemb, size_t size, int
              (*compare) (const void *_key, const void *_base));
```

Parameters

base	Pointer to the array to sort.
nmemb	Dimension of the array pointed to by *base*.
size	Size of the array elements.
compare	The comparison function, which takes two arguments and returns:

<0 (negative value)	if *_key* is less than *_base*
0	if *_key* equals *_base*
>0 (positive value)	if *_key* is greater than *_base*

Return value

None.

Description

Sorts an array of *nmemb* objects pointed to by *base*.

Header file

`stdlib.h`

---

rand `int rand(void)`

Parameters

None.

Return value

The next `int` in the random number sequence.

Description

Computes the next in the current sequence of pseudo-random integers, converted to lie in the range [`0,RAND_MAX`].

See *srand*, page 38`,` for a description of how to seed the pseudo-random sequence.

Header file

`stdlib.h`

---

realloc `void *realloc(void *ptr, size_t size)`

Parameters

ptr	A pointer to the start of the memory block.
size	A value of type `size_t` specifying the size of the object.

Return value

Result	Value
Successful	A pointer to the start (lowest address) of the memory block.
Unsuccessful	Null, if no memory block of the required size or greater was available.

Table 15: realloc return values

Description

Changes the *`size`* of a memory block (which must be allocated by `malloc`, `calloc`, or `realloc`).

Header file

`stdlib.h`

---

scanf `int scanf(const char *format, …)`

Parameters

format	A pointer to a format string.
…	Optional pointers to the variables that are to receive values.

Return value

Result	Value
Successful	The number of successful conversions.
Unsuccessful	-1 if the input was exhausted.

Table 16: scanf return values

Description

Reads formatted data from standard input.

Since a complete formatter demands a lot of space there are several different formatters to choose between. For more information, see the *IAR C Compiler Reference Guide*.

The parameter *format* is a string consisting of a sequence of ordinary characters and conversion specifications. Each ordinary character reads a matching character from the input. Each conversion specification accepts input meeting the specification, converts it, and assigns it to the object pointed to by the next successive argument following *format*.

If the format string contains white-space characters, input is scanned until a non-white-space character is found.

The form of a conversion specification is as follows:

```
% [assign_suppress] [field_width] [length_modifier]
conversion
```

Items inside `[ ]` are optional.

Assign suppress

If a `*` is included in this position, the field is scanned but no assignment is carried out.

field_width

The *field_width* is the maximum field to be scanned. The default is until no match occurs.

length_modifier

The effect of each *length_modifier* is as follows:

Length modifier	Before	Meaning
l	d, i, or n	long int as opposed to int.
	o, u, or x	unsigned long int as opposed to unsigned int.
	e, E, g, G, or f	double operand as opposed to *float*.
h	d, i, or n	short int as opposed to int.
	o, u, or x	unsigned short int as opposed to unsigned int.
L	e, E, g, G, or f	long double operand as opposed to float.

Table 17: scanf length modifier

Conversion

The meaning of each conversion is as follows:

Conversion	Meaning
d	Optionally signed decimal integer value.
i	Optionally signed integer value in standard C notation, that is, is decimal, octal (0n) or hexadecimal (0xn, 0Xn).
o	Optionally signed octal integer.
u	Unsigned decimal integer.
x	Optionally signed hexadecimal integer.
X	Optionally signed hexadecimal integer (equivalent to x).
f	Floating-point constant.
e E g G	Floating-point constant (equivalent to f).
s	Character string.
c	One or field_width characters.
n	No read, but store number of characters read so far in the integer pointed to by the next argument.
p	Pointer value (address).

Table 18: scanf conversion

Conversion	Meaning
[	Any number of characters matching any of the characters before the terminating]. For example, [abc] means a, b, or c.
[]	Any number of characters matching] or any of the characters before the further, terminating]. For example, []abc] means], a, b, or c.
[^	Any number of characters not matching any of the characters before the terminating]. For example, [^abc] means not a, b, or c.
[^]	Any number of characters not matching] or any of the characters before the further, terminating]. For example, [^]abc] means not], a, b, or c.
%	% character.

Table 18: scanf conversion

In all conversions except c, n, and all varieties of [, leading white-space characters are skipped.

scanf indirectly calls getchar, which must be adapted for the actual target hardware configuration.

Header file

stdio.h

Examples

For example, after the following program:

```
int n, i;
char name[50];
float x;
n = scanf("%d%f%s", &i, &x, name)
```

this input line:

```
25 54.32E-1 Hello World
```

will set the variables as follows:

```
n = 3, i = 25, x = 5.432, name="Hello World"
```

and this function:

```
scanf("%2d%f%*d %[0123456789]", &i, &x, name)
```

with this input line:

```
56789 0123 56a72
```

will set the variables as follows:

```
i = 56, x = 789.0, name="56" (0123 unassigned)
```

setjmp `int setjmp(jmp_buf env)`

Parameters

env	An object of type `jmp_buf` into which `setjmp` is to store the environment.

Return value

Zero.

Execution of a corresponding `longjmp` causes execution to continue as if it was a return from `setjmp`, in which case the value of the `int` value given in the `longjmp` is returned.

Description

Sets up a jump return point.

Saves the environment in *env* for later use by `longjmp`.

Note: `setjmp` must always be used in the same function or at a higher nesting level than the corresponding call to `longjmp`.

Header file

`setjmp.h`

sin `double sin(double arg)`

Parameters

arg	A `double` value in radians.

Return value

The `double` sine of *arg*.

Description

Computes the sine of a number.

Header file

`math.h`

sinh `double sinh(double *arg*)`

Parameters

arg	A `double` value in radians.

Return value

The `double` hyperbolic sine of *arg*.

Description

Computes the hyperbolic sine of *arg* radians.

Header file

`math.h`

sprintf `int sprintf(char *s, const char *format, …)`

Parameters

s	A pointer to the string that is to receive the formatted data.
format	A pointer to the format string.
…	The optional values that are to be printed under the control of *format*.

Return value

Result	Value
Successful	The number of characters written.
Unsuccessful	A negative value if an error occurred.

Table 19: sprintf return values

Description

Writes formatted data to a string.

Operates exactly as `printf` except that the output is directed to a string. See *printf*, page 26, for details.

`sprintf` does not use the function `putchar`, and therefore can be used even if `putchar` is not available for the target configuration.

Since a complete formatter demands a lot of space there are several different formatters to choose between. For more information, see the *IAR C Compiler Reference Guide*.

Header file

`stdio.h`

`sqrt` `double sqrt(double *arg*)`

Parameters

arg	A `double` value.

Return value

The `double` square root of *arg*.

Description

Computes the square root of a number.

Header file

`math.h`

`srand` `void srand(unsigned int *seed*)`

Parameters

seed	An `unsigned int` value identifying the particular random number sequence.

Return value

None.

Description

Selects a repeatable sequence of pseudo-random numbers.

The function `rand` is used to get successive random numbers from the sequence. If `rand` is called before any calls to `srand` have been made, the sequence generated is that which is generated after `srand(1)`.

Header file

`stdlib.h`

---

sscanf `int sscanf(const char *s, const char *format, …)`

Parameters

s	A pointer to the string containing the data.
format	A pointer to a format string.
…	Optional pointers to the variables that are to receive values.

Return value

Result	Value
Successful	The number of characters written.
Unsuccessful	A negative value if an error occurred.

Table 20: sscanf return values

Description

Reads formatted data from a string.

Operates exactly as `scanf` except the input is taken from the string *s*. See `scanf` for details.

The function `sscanf` does not use `getchar`, and so can be used even when `getchar` is not available for the target configuration.

Since a complete formatter demands a lot of space there are several different formatters to choose. For more information, see the *IAR C Compiler Reference Guide*.

Header file

`stdio.h`

---

strcat `char *strcat(char *s1, const char *s2)`

Parameters

s1	A pointer to the first string.
s2	A pointer to the second string.

Return value

s1.

Description

Concatenates strings by appending a copy of the second string to the end of the first string. The initial character of the second string overwrites the terminating null character of the first string.

Header file

`string.h`

---

strchr `char *strchr(const char *s, int c)`

Parameters

c	An `int` representation of a character.
s	A pointer to a string.

Return value

If successful, a pointer to the first occurrence of *c* (converted to a `char`) in the string pointed to by *s*.

If unsuccessful due to *c* not being found, null.

Description

Searches for the first occurrence of a character (converted to a `char`) in a string. The terminating null character is considered to be part of the string.

Header file

`string.h`

---

strcmp `int strcmp(const char *s1, const char *s2)`

Parameters

s1	A pointer to the first string.
s2	A pointer to the second string.

Return value

The `int` result of comparing the two strings:

Return value	Meaning
`>0`	`s1 > s2`
`=0`	`s1 = s2`
`<0`	`s1 < s2`

Table 21: strcmp return values

Description

Compares two strings.

Header file

`string.h`

`strcoll` `int strcoll(const char *s1, const char *s2)`

Parameters

s1	A pointer to the first string.
s2	A pointer to the second string.

Return value

The `int` result of comparing the two strings:

Return value	Meaning
`>0`	`s1 > s2`
`=0`	`s1 = s2`
`<0`	`s1 < s2`

Table 22: strcoll return values

Description

Compares two strings. This function operates identically to `strcmp` and is provided for compatibility only.

Header file

`string.h`

strcpy `char *strcpy(char *s1, const char *s2)`

Parameters

s1	A pointer to the destination object.
s2	A pointer to the source string.

Return value

s1.

Description

Copies a string into an object.

Header file

`string.h`

strcspn `size_t strcspn(const char *s1, const char *s2)`

Parameters

s1	A pointer to the subject string.
s2	A pointer to the object string.

Return value

The `int` length of the maximum initial segment of the string pointed to by *s1* that consists entirely of characters *not* from the string pointed to by *s2*.

Description

Spans excluded characters in string.

Finds the maximum initial segment of a subject string that consists entirely of characters *not* from an object string.

Header file

`string.h`

strerror `char * strerror (int errnum)`

Parameters

errnum	The error message to return.

Return value

The function returns the following strings.

errnum	String returned
`EZERO`	`"no error"`
`EDOM`	`"domain error"`
`ERANGE`	`"range error"`
`errnum < 0 || errnum > Max_err_num`	`"unknown error"`
All other numbers	`"error No.` *errnum*`"`

Table 23: strerror return values

Description

Returns an error message string.

Header file

`string.h`

strlen `size_t strlen(const char *s)`

Parameters

s	A pointer to a string.

Return value

An object of type `size_t` indicating the length of the string.

Description

Finds the number of characters in a string, not including the terminating null character.

Header file

`string.h`

strncat string.h

Declaration

`char *strncat(char *s1, const char *s2, size_t n)`

Parameters

s1	A pointer to the destination string.
s2	A pointer to the source string.
n	The number of characters of the source string to use.

Return value

s1.

Description

Concatenates a specified number of characters with a string by appending not more than *n* initial characters from the source string to the end of the destination string.

Header file

string.h

---

strncmp `int strncmp(const char *s1, const char *s2, size_t n)`

Parameters

s1	A pointer to the first string.
s2	A pointer to the second string.
n	The number of characters of the source string to compare.

Return value

The `int` result of the comparison of not more than *n* initial characters of the two strings:

Return value	Meaning
>0	s1 > s2
=0	s1 = s2

Table 24: strncmp return values

Return value	Meaning
<0	s1 < s2

Table 24: strncmp return values

Description

Compares not more than *n* initial characters of two strings.

Header file

`string.h`

strncpy `char *strncpy(char *s1, const char *s2, size_t n)`

Parameters

s1	A pointer to the destination object.
s2	A pointer to the source string.
n	The number of characters of the source string to copy.

Return value

s1.

Description

Copies not more than *n* initial characters from the source string into the destination object.

Header file

`string.h`

strpbrk `char *strpbrk(const char *s1, const char *s2)`

Parameters

s1	A pointer to the subject string.
s2	A pointer to the object string.

Return value

Result	Value
Successful	A pointer to the first occurrence in the subject string of any character from the object string.
Unsuccessful	Null if none were found.

Table 25: strpbrk return values

Description

Searches one string for any occurrence of any character from a second string.

Header file

`string.h`

`strrchr` `char *strrchr(const char *s, int c)`

Parameters

s	A pointer to a string.
c	An `int` representing a character.

Return value

If successful, a pointer to the last occurrence of *c* in the string pointed to by *s*.

Description

Finds character from right of string by searching for the last occurrence of a character (converted to a `char`) in a string. The terminating null character is considered to be part of the string.

Header file

`string.h`

`strspn` `size_t strspn(const char *s1, const char *s2)`

Parameters

s1	A pointer to the subject string.
s2	A pointer to the object string.

Return value

The length of the maximum initial segment of the string pointed to by *s1* that consists entirely of characters from the string pointed to by *s2*.

Description

Spans characters in a string by finding the maximum initial segment of a subject string that consists entirely of characters from an object string.

Header file

```
string.h
```

strstr `char *strstr(const char *s1, const char *s2)`

Parameters

s1	A pointer to the subject string.
s2	A pointer to the object string.

Return value

Result	Value
Successful	A pointer to 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*.
Unsuccessful	Null if the string was not found. *s1* if *s2* is pointing to a string with zero length.

Table 26: strstr return values

Description

Searches one string for an occurrence of a second string (a substring).

Header file

```
string.h
```

strtod `double strtod(const char *nptr, char **endptr)`

Parameters

nptr	A pointer to a string.
endptr	A pointer to a pointer to a string.

Return value

Result	Value
Successful	The `double` result of converting the ASCII representation of an floating-point constant in the string pointed to by *nptr*, leaving *endptr* pointing to the first character after the constant.
Unsuccessful	Zero, leaving *endptr* indicating the first non-space character.

Table 27: strtod return values

Description

Converts a string (the ASCII representation of a number) into a `double`, stripping any leading white space.

Header file

`stdlib.h`

strtok `char *strtok(char *s1, const char *s2)`

Parameters

s1	A pointer to a string to be broken into tokens.
s2	A pointer to a string of delimiters.

Return value

Result	Value
Successful	A pointer to the token.
Unsuccessful	Zero.

Table 28: strtok return values

Description

Breaks a string into tokens by finding the next token in the string *s1*, separated by one or more characters from the string of delimiters *s2*.

The first time you call strtok, *s1* should be the string you want to break into tokens. strtok saves this string. On each subsequent call, *s1* should be NULL. strtok searches for the next token in the string it saved. *s2* can be different from call to call.

If strtok finds a token, it returns a pointer to the first character in it. Otherwise it returns NULL. If the token is not at the end of the string, strtok replaces the delimiter with a null character (\0).

Header file

string.h

strtol `long int strtol(const char *nptr, char **endptr, int base)`

Parameters

nptr	A pointer to a string.
endptr	A pointer to a pointer to a string.
base	An int value specifying the base.

Return value

Result	Value
Successful	The long int result of converting the ASCII representation of an integer constant in the string pointed to by *nptr*, leaving *endptr* pointing to the first character after the constant.
Unsuccessful	Zero, leaving *endptr* indicating the first non-space character.

Table 29: strtol return values

Description

Converts a string (the ASCII representation of a number) into a long int using the specified base, and stripping any leading white space.

If the base is zero the sequence expected is an ordinary integer. Otherwise the expected sequence consists of digits and letters representing an integer with the radix specified by *base* (must be between 2 and 36). The letters [a,z] and [A,Z] are ascribed the values 10 to 35. If the base is 16, the 0x portion of a hex integer is allowed as the initial sequence.

Header file

`stdlib.h`

strtoul

```
unsigned long int strtoul(const char *nptr, char **endptr,
       base int)
```

Parameters

nptr	A pointer to a string.
endptr	A pointer to a pointer to a string.
base	An `int` value specifying the base.

Return value

Result	Value
Successful	The `unsigned long int` result of converting the ASCII representation of an integer constant in the string pointed to by *nptr*, leaving *endptr* pointing to the first character after the constant.
Unsuccessful	Zero, leaving *endptr* indicating the first non-space character.

Table 30: strtoul return values

Description

Converts a string (the ASCII representation of a number) into an `unsigned long int` using the specified base, stripping any leading white space.

If the base is zero the sequence expected is an ordinary integer. Otherwise the expected sequence consists of digits and letters representing an integer with the radix specified by *base* (must be between `2` and `36`). The letters `[a,z]` and `[A,Z]` are ascribed the values `10` to `35`. If the base is `16`, the `0x` portion of a hex integer is allowed as the initial sequence.

Header file

`stdlib.h`

strxfrm `size_t strxfrm(char *s1, const char *s2, size_t n)`

Parameters

s1	Return location of the transformed string.
s2	String to transform.
n	Maximum number of characters to be placed in *s1*.

Return value

The length of the transformed string, not including the terminating null character.

Description

Transforms a string and returns the length.

The transformation is such that if the `strcmp` function is applied to two transformed strings, it returns a value corresponding to the result of the `strcoll` function applied to the same two original strings.

Header file

`string.h`

tan `double tan(double arg)`

Parameters

arg	A `double` value in radians.

Return value

The `double` tangent of *arg*.

Description

Computes the tangent of *arg* radians.

Header file

`math.h`

tanh `double tanh(double arg)`

Parameters

arg	A `double` value in radians.

Return value

The `double` hyperbolic tangent of *arg*.

Description

Computes the hyperbolic tangent of *arg* radians.

Header file

`math.h`

tolower `int tolower(int c)`

Parameters

c	The `int` representation of a character.

Return value

The `int` representation of the lower case character corresponding to *c*.

Description

Converts a character into lower case.

Header file

`ctype.h`

toupper `int toupper(int c)`

Parameters

c	The `int` representation of a character.

Return value

The `int` representation of the upper case character corresponding to *c*.

Description

Converts a character into upper case.

Header file

`ctype.h`

`va_arg` `type va_arg(va_list` *`ap`*`,` *`mode`*`)`

Parameters

ap	A value of type `va_list`.
mode	A type name such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a `*` to `type`.

Return value

See below.

Description

Expands to the next argument in a function call.

A macro that expands to an expression with the type and value of the next argument in the function call. After initialization by `va_start`, this is the argument after that specified by `parmN`. `va_arg` advances *`ap`* to deliver successive arguments in order.

For an example of the use of `va_arg` and associated macros, see the files `printf.c` and `intwri.c`.

Header file

`stdarg.h`

`va_end` `void va_end(va_list` *`ap`*`)`

Parameters

ap	A pointer of type `va_list` to the variable-argument list.

Return value

See below.

Description

Ends reading function call arguments.

A macro that facilitates normal return from the function whose variable argument list was referenced by the expansion `va_start` that initialized `va_list ap`.

Header file

`stdarg.h`

`va_list` `char *va_list[1]`

Parameters

None.

Return value

See below.

Description

Argument list type.

An array type suitable for holding information needed by `va_arg` and `va_end`.

Header file

`stdarg.h`

`va_start` `void va_start(va_list` *`ap`*`,` *`parmN`*`)`

Parameters

ap	A pointer of type `va_list` to the variable-argument list.
parmN	The identifier of the rightmost parameter in the variable parameter list in the function definition.

Return value

See below.

Description

Starts reading function call arguments.

A macro that initializes *ap* for use by `va_arg` and `va_end`.

Header file

`stdarg.h`

---

vprintf `int vprintf(const char * format, va_list argptr)`

Parameters

format	A pointer to the format string.
argptr	List of arguments.

Return value

Result	Value
Successful	The number of characters written.
Unsuccessful	A negative value, if an error occurred.

Table 31: vprintf return values

Description

Writes formatted data to standard output; performs the same function as `printf`, but accepts a pointer to a list of arguments rather than the arguments themselves. For format details, see *printf*, page 26, and for argument list details, see *va_list*, page 54.

Header file

`stdio.h`

---

vsprintf `int vsprintf(char * s, const char * format, va_list argptr)`

Parameters

s	A pointer to the string that is to receive the formatted data.
format	A pointer to the format string.
argptr	List of arguments.

Return value

Result	Value
Successful	The number of characters written.
Unsuccessful	A negative value, if an error occurred.

Table 32: vsprintf return values

Description

Writes formatted data to a buffer; performs the same function as `sprintf`, but accepts a pointer to a list of arguments rather than the arguments themselves. For details of *`s`* and *`format`*, see *sprintf*, page 37, and for argument list details, see *va_list*, page 54.

Header file

`stdio.h`

_formatted_read

```
int _formatted_read (const char **line, const char **format,
       va_list ap)
```

Parameters

line	A pointer to a pointer to the data to scan.
format	A pointer to a pointer to a standard `scanf` format specification string.
ap	A pointer of type `va_list` to the variable argument list.

Return value

The number of successful conversions.

Description

Reads formatted data. This function is the basic formatter of `scanf`.

`_formatted_read` is concurrently reusable (reentrant).

Note: The use of `_formatted_read` requires the special ANSI-defined macros in the file `stdarg.h`, described above. In particular:

- There must be a variable *ap* of type `va_list`.
- There must be a call to `va_start` before calling `_formatted_read`.
- There must be a call to `va_end` before leaving the current context.

- The argument to va_start must be the formal parameter immediately to the left of the variable argument list.

Header file

icclbutl.h

_formatted_write

```
int _formatted_write (const char *format, void outputf (char,
       void *), void *sp, va_list ap)
```

Parameters

format	A pointer to standard printf/sprintf format specification string.
outputf	A function pointer to a routine that actually writes a single character created by _formatted_write. The first parameter to this function contains the actual character value and the second a pointer whose value is always equivalent to the third parameter of _formatted_write.
sp	A pointer to some type of data structure that the low-level output function may need. If there is no need for anything more than just the character value, this parameter must still be specified with (void *) 0 as well as declared in the output function.
ap	A pointer of type va_list to the variable-argument list.

Return value

The number of characters written.

Description

Formats and writes data. This function is the basic formatter of printf and sprintf, but through its universal interface can easily be adapted for writing to non-standard display devices.

Since a complete formatter demands a lot of space there are several different formatters to choose. For more information, see the *IAR C Compiler Reference Guide*.

_formatted_write is concurrently reusable (reentrant).

Note: The use of _formatted_write requires the special ANSI-defined macros in the file stdarg.h, described above. In particular:

- There must be a variable *ap* of type va_list.
- There must be a call to va_start before calling _formatted_write.
- There must be a call to va_end before leaving the current context.
- The argument to va_start must be the formal parameter immediately to the left of the variable argument list.

For an example of how to use _formatted_write, see the file printf.c.

Header file

icclbutl.h

_medium_read

```
int _medium_read (const char **line, const char **format,
       va_list ap)
```

Parameters

line	A pointer to a pointer to the data to scan.
format	A pointer to a pointer to a standard scanf format specification string.
ap	A pointer of type va_list to the variable argument list.

Return value

The number of successful conversions.

Description

Reads formatted data excluding floating-point numbers.. This is a reduced version of _formatted_read which is half the size.

For further information see *_formatted_read*, page 56.

Header file

icclbutl.h

_medium_write

```
int _medium_write (const char *format, void outputf(char,
       void *), void *sp, va_list ap)
```

Parameters

format	A pointer to standard printf/sprintf format specification string.

outputf	A function pointer to a routine that actually writes a single character created by `_formatted_write`. The first parameter to this function contains the actual character value and the second a pointer whose value is always equivalent to the third parameter of `_formatted_write`.
sp	A pointer to some type of data structure that the low-level output function may need. If there is no need for anything more than just the character value, this parameter must still be specified with `(void *) 0` as well as declared in the output function.
ap	A pointer of type `va_list` to the variable-argument list.

Return value

The number of characters written.

Description

Writes formatted data excluding floating-point numbers. This is a reduced version of `_formatted_write` which is half the size.

For further information see *_formatted_write*, page 57.

Header file

`icclbutl.h`

`_small_write`

```
int _small_write (const char *format, void outputf(char, void
       *), void *sp, va_list ap)
```

Parameters

format	A pointer to standard `printf`/`sprintf` format specification string.
outputf	A function pointer to a routine that actually writes a single character created by `_formatted_write`. The first parameter to this function contains the actual character value and the second a pointer whose value is always equivalent to the third parameter of `_formatted_write`.

sp	A pointer to some type of data structure that the low-level output function may need. If there is no need for anything more than just the character value, this parameter must still be specified with `(void *) 0` as well as declared in the output function.
ap	A pointer of type `va_list` to the variable-argument list.

Return value

The number of characters written.

Description

This is a small version of `_formatted_write` that is about a quarter of the size. It supports only the following specifiers for int objects:

`%%`, `%d`, `%o`, `%c`, `%s`, and `%x`

It does not support field width or precision arguments, and no diagnostics will be produced if unsupported specifiers or modifiers are used. For further information see *_formatted_write*, page 57.

Header file

`icclbutl.h`