Common C Library Functions

<math.h>

Function	Argument Type	Description	Result Type
ceil(x)	double	Return the smallest **`double`** larger than or equal to **`x`** that can be represented as an **`int`**.	double
floor(x)	double	Return the largest **`double`** smaller than or equal to **`x`** that can be represented as an **`int`**.	double
abs(x)	int	Return the absolute value of **`x`**, where **`x`** is an **`int`**.	int
fabs(x)	double	Return the absolute value of **`x`**, where **`x`** is a floating point number.	double
sqrt(x)	double	Return the square root of **`x`**, where **`x`** >= 0.	double
pow(x,y)	double x, double y	Return x to the y power, x^y.	double
cos(x)	double	Return the cosine of **`x`**, where **`x`** is in radians.	double
sin(x)	double	Return the sine of **`x`**, where **`x`** is in radians.	double
tan(x)	double	Return the tangent of **`x`**, where **`x`** is in radians.	double
exp(x)	double	Return the exponential of **`x`** with the base `e`, where `e` is 2.718282.	double
log(x)	double	Return the natural logarithm of **`x`**.	double
log10(x)	double	Return the base 10 logarithm of **`x`**.	double

<string.h>

Function	Meaning	Description	Function Header
`strcat()`	string concatenation	appends one string to another	`char *strcat(char *dest, const char *src)` Appends the string pointed to, by *src* to the end of the string pointed to by *dest*.
`strncat()`	string concatenation of n characters	appends a portion of a string to another string	`char *strncat(char *dest, const char *src, size_t n)` Appends the string pointed to, by *src* to the end of the string pointed to, by *dest* up to n characters long.

`strchr()`	string has character	finds the first occurrence of a specified character in a string	`char *strchr(const char *str, int c)` Searches for the first occurrence of the character c (an unsigned char) in the string pointed to, by the argument *str*.
`strrchr()`	string has character (search from end)	finds the last occurrence of a specified character in a string	`char *strrchr(const char *str, int c)` Searches for the last occurrence of the character c (an unsigned char) in the string pointed to by the argument *str*.
`strstr()`	string has substring	finds the position of the first character of one string in another	`char *strstr(const char *haystack, const char *needle)` Finds the first occurrence of the entire string *needle* (not including the terminating null character) which appears in the string *haystack*.
`strcmp()`	string comparison of n characters	compares two strings	`int strcmp(const char *str1, const char *str2)` Compares the string pointed to, by *str1* to the string pointed to by *str2*.
`strncmp()`	string comparison of n characters	compares two strings up to a specified number of characters	`int strncmp(const char *str1, const char *str2, size_t n)` Compares at most the first n bytes of *str1* and *str2*.
`strcpy()`	string copy	copies a string to an array	`char *strcpy(char *dest, const char *src)` Copies the string pointed to, by *src* to *dest*.
`strncpy()`	string copy of n characters	copies a portion of a string to an array	`char *strncpy(char *dest, const char *src, size_t n)` Copies up to n characters from the string pointed to, by *src* to *dest*.
`strpbrk()`	string pointer break	finds the first occurrence of any specified characters in a substring within a string	`char *strpbrk(const char *str1, const char *str2)` Finds the first character in the string *str1* that matches any character specified in *str2*.
`strlen()`	string length	computes the length of a string	`size_t strlen(const char *str)` Computes the length of the string str up to but not including the terminating null character

<ctype.h>

Function	Description: Returns True if Argument is
int isalnum(int c)	alphanumeric (alphabetic or numeric), i.e. 'A' - 'Z', 'a' - 'z' or '0' - '9'
int isalpha(int c)	alphabetic, i.e. 'A' - 'z' or 'a' - 'z'
int iscntrl(int c)	a control character, e.g. Control-B
int isdigit(int c)	a digit, i.e. '0' - '9'
int isgraph(int c)	any printing character other than a space, i.e. ASCII 33 – 127
int islower(int c)	a lowercase character, i.e. 'a' - 'z'
int isprint(int c)	a printing character, i.e. ASCII 32 –127
int ispunct(int c)	a punctuation character
int isspace(int c)	a whitespace character, e.g. space, newline, formfeed, carriage return, i.e. ASCII 9 - 13 or 32
int isupper(int c)	an uppercase character, i.e. 'A' - 'Z'
int isxdigt(int c)	a hexadecimal digit character, i.e. '0' - '9', 'A' - 'F', 'a' - 'f'

Conversion Function	Description
int tolower(int c)	This function converts uppercase letters to lowercase.
int toupper(int c)	This function converts lowercase letters to uppercase.

<stdio.h>

Function	Description
scanf()	`int scanf(const char *format, ...)` Reads formatted input from stdin.
printf()	`int printf(const char *format, ...)` Sends formatted output to stdout.
getchar()	`int getchar(void)` Gets a character (an unsigned char) from stdin.

putchar()	`int putchar(int char)` Writes a character (an unsigned char) specified by the argument char to stdout.
fgets()	`char *fgets(char *str, int n, FILE *stream)` Reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.
puts()	`int puts(const char *str)` Writes a string to stdout up to but not including the null character. A newline character is appended to the output.

sscanf()	`int sscanf(const char *str, const char *format, ...)`
	`Reads formatted input from a string.`

fprintf()	`int fprintf(FILE *stream, const char *format, ...)`
	`Sends formatted output to a stream.`

<stdlib.h>

Function	**Description**
atof()	`double atof(const char *str)` Converts the string pointed to, by the argument *str* to a floating-point number (type double).
atoi()	`int atoi(const char *str)` Converts the string pointed to, by the argument *str* to an integer (type int).

Operator Precedence Table in C

Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Cast (convert value to temporary value of *type*) Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
| |	Logical OR	left-to-right
? :	Ternary conditional	right-to-left

=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL