Note:
To change the product logo for your own print manual or PDF, click "Tools > Manual Designer" and modify the print manual template.

Scientific formula calculator

Title page 1

Use this page to introduce the product

by Enter your company name

This is "Title Page 1" - you may use this page to introduce your product, show title, author, copyright, company logos, etc.

This page intentionally starts on an odd page, so that it is on the right half of an open book from the readers point of view. This is the reason why the previous page was blank (the previous page is the back side of the cover)

Scientific formula calculator

Printed: November 2025 in (whereever you are located)

Publisher

...enter name...

Managing Editor

...enter name...

Technical Editors

...enter name...
...enter name...

Cover Designer

...enter name...

Team Coordinator

...enter name...

Production

...enter name...

Special thanks to:

All the people who contributed to this document, to mum and dad and grandpa, to my sisters and brothers and mothers in law, to our secretary Kathrin, to the graphic artist who created this great product logo on the cover page (sorry, don't remember your name at the moment but you did a great work), to the pizza service down the street (your daily Capricciosas saved our lives), to the copy shop where this document will be duplicated, and and and...

Last not least, we want to thank EC Software who wrote this great help tool called HELP & MANUAL which printed this document.

Table of Contents

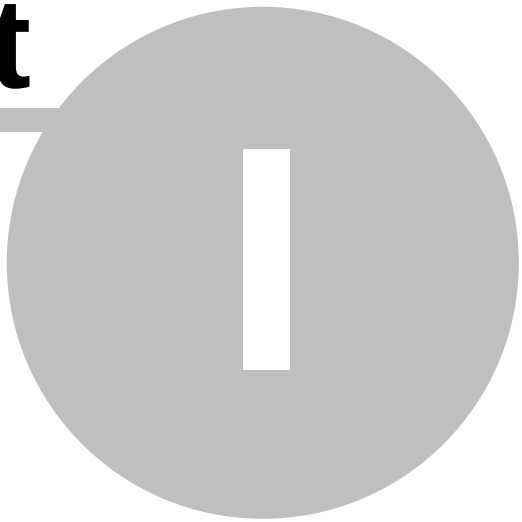
Foreword

This is just another title page
placed between table of contents
and topics

Top Level Intro

This page is printed before a new
top-level chapter starts

Part



1 Introduction

Input for this calculator are normal C expressions containing operators, float or integer constants, variables.
Precedence and semantic of operators is the same as in C language. Variables and functions names are case
insensitive. Following functions and operators are supported:

Functions:		**Operators:**			
abs(x|z)	Absolute value| Complex	**Common**			
sin(x|z)	Sin of radian|Complex	**++**	Increment	**--**	Decrement
sing(x)	Sin of degrees	**+**[1]	Plus (also for string)	**-**[1]	Minus
cos(x|z)	Cos of radian|Complex	**!**	Not	**~**[2]	Complement
cosg(x)	Cos of degrees	*****[1]	Multiplication	**/**[1]	Division
tg(x|z), **tan**(x|z)	Tg of radian|Complex	**%**	Remainder	******[1]	Power
tgg(x)	Tg of degrees	**&&**	Logical AND	**&**	AND
ctg(x|z),**cot**(x|z)	Ctg of radian|Complex	**||**	Logical OR	**|**	OR
ctgg(x)	Ctg of degrees	**<<**	Shift left	**>>**	Shift right
arcsin(x|z), **asin**(x|z)	ArcSin in radian| Complex	**>>>**	Unsign shift right	**#**	XOR
arccos(x|z), **acos**(x|z)	ArcCos in radian| Complex	**<**	Less	**>**	Greater
arctg(x|z), **atan**(x|z)	ArcTg in radian| Complex	**!=**[1]	Not equal	**<>**[1]	Not equal
arcctg(x|z), **acot**(x|z)	ArcCtg in radian| Complex	**+=**	Increase	**-=**	Decrease
sinh(x|z), **sh**(x|z)	Hyperbolic sine	***=**	Mul and assign	**/=**	Div and assign
cosh(x|z), **ch**(x|z)	Hyperbolic cosine	**%=**	Rem and assign	****=**	Pow and assign
tanh(x|z), **th**(x|z)	Hyperbolic tangent	**<<=**	Shift and assign	**>>=**	Shift and assign
ctanh(x|z), **cth**(x|z)	Hyperbolic cotangent	**#=**	XOR and assign	**&=**	AND and assign
asinh(x|z), **arsh**(x|z)	Hyperbolic arcsine	**^=**	XOR and assign	**>>>=**	Shift and assign
acosh(x|z), **arch**(x|z)	Hyperbolic arccosine	**'Pascal' - style**			
atanh(x|z), **arth**(x|z)	Hyperbolic arctangent	**:=**[1]	Assign	**=**[1]	Compare
acoth(x|z), **arcth**(x|z)	Hyperbolic arccotangent	**^**[1]	Power	**//**[1]	Parallel resistors
atan2(y,x)	Arc tangent of y/x.	**'C' - style**			
sqrt(x|z), **root2**(x|z)	Square root	**=**[1]	Assign	**==**[1]	Compare
exp(x|z)	Exponent, e^x	**^**	XOR	**//**[1]	Parallel

					resistors
ln(x|z), **log**(x|z)	Natural logarithm	*[1]Also for complex operands*			
exp10(x)	10^x	*[2]Complex conjugate: ~(1+2i)=1-2i*			
lg(x), **log10**(x)	Logarithm base 10, $Log_{10}(x)$	*First using of '==' switches engine to 'C'-style syntax.*			
root3(x), **cbrt**(x)	Cube root, $x^{1/3}$	*First using of ':=' switches engine to 'Pas'-style syntax.*			
rootn(x|z,n|z)	n^{th} root, $x^{1/n}$	*Default is 'Pas'-style syntax.*			
log2(x)	Logarithm base 2, $Log_2(x)$	**printf (fmt, ...)**	C-style *printf*. Result in the string value.		
int(x)	Convert to int	**fprn(fmt, ...)**			
float(n)	Convert to float	**prn(fmt, ...)**			
round(x)	Round value	**datatime**("2017.11.23 22:00:20")		Return Unix time	
frac(x)	Returns the fractional part	**timezone**			
not(n)	Inversion all bits, **~**n	**daylight**			
erf(x)	Error function	**tz**	Offset from UTC in hours		
erfc(x)	Complementary error function	**Examples:**			
now(n)	Current Unix time (+/- n hours)	L:=130u;c:=2.2n;f:=1/(2*pi*sqrt(l*c))		297.6k	
rnd(x)	Random value 0..x	72-20%		57.6	
hypot(x,y)	Hypotenuse	72-20%+5%		60.48	
db(x)	Decibel, 10*lg(x)	500*5%		12500	
np(x)	Neper, 20*lg(x)	1/2%		50	
adb(x)	$10^{x/10}$	57.6%%72		-20	
anp(x)	$10^{x/20}$	abs(3+4j)		5	
pow(x|z,y|z)	x^y	z:=3+4i		3+4i	
logn(x|z,y|z)	$Log_x(y)$	*The last use of the imaginary part notation (i, j) is used to print the result.*			
Vout(Vref,Rh,Rl)	Vout=Vref*(Rh+Rl)/Rl				
ee(n,x)	E series of preferred numbers ee(24, 5k)=5.1k				
cs(x)	Cross section of diameter				
acs(x)	Diameter of cross section				
awg(n)	American wire gauge to mm	**Hotkeyes:**			
sawg(n)	American wire gauge to mm^2	Ctrl+N	New calculation		
aawg(x)	mm to american wire gauge	Shift+Ctrl++/-	Opacity		
swg(n)	British standard wire gauge to mm	Ctrl++/-	Font size		
sswg(n)	British standard wire gauge to mm^2	Ctrl+R	Root2(...)		
aswg(x)	mm to british standard wire gauge	Ctrl+S	(...)^2		
cmplx(x,y)	Return complex value	Ctrl+I	1/(...)		
gcd(n,m)	Greatest common divisor of n and	Ctrl+[	(...)		

	m			
invmod(n,m)	Inverse of n modulo m		Ctrl+]	(...)
re(z)	Return real value of complex		Ctrl+Home	Show calculator
im(z)	Return imaginary value	**Data formats:**		
pol(z)	atan(im(z)/re(z))	Hex	0x1000, $1000	4096
min(x,y)	Minimum of x, y	Octal	01000, 0o1000	512
max(x,y)	Maximum of x, y	Binary	0b1000	8
fact(n), n**!**	Factorial (n!)	Degrees	10`20'40"	0.180544 rad.
prime(n)	Smallest prime number	Date time	1:c1:y1:d1:h1:m1:s	189377247661s
Constants:		Char	'c'	'c'
version	2.02 or above	WChar	L'c', 'c'W	'c'W
pi	3.141592653589	String	'string'+ " value"	"String value"
e	2.718281828459	Other	\377 = 255, \e = 27, \n = 10, \xff = 255	
phi	1.618033988749 (Golden ratio)	Normalized	1.8e6	1800000
Special:		Scientific	1.8M	1800000
help(0**)**	Call help	Engineering	1k33	1330
opacity(100**)**	Set opacity level in percent	Computing	1KB	1024
binwide(32**)**	Set binary width	Fractional	3+9/64	Pi
menu(0**)**	Hide (0) or show (1) menu.	Gauge	#28	awg(28) =0.3211265489669004
vars(0**)**	Show variables list	Inch	2"5	0.0635=2+1/2 "
font(x**)**	Set calculator font size.	Complex	3+4i	3+4i

Range postfix:									
Binary (power of 2)									
1**YB**	yotta	1208925819614629174706176	2^{80}	1**ZB**	zetta	1180591620717411303424	2^{70}		
1**EB**	exa	1152921504606846976	2^{60}	1**PB**	peta	1125899906842624	2^{50}		
1**TB**	tera	1099511627776	2^{40}	1**GB**	giga	1073741824	2^{30}		
1**MB**	mega	1048576	2^{20}	1**KB**	kilo	1024	2^{10}		
Decimal (power of 10)									
				EU	US				
1**Y**	yotta		10^{24}	Quadrillion		1**y**	yocto		10^{-24}
1**Z**	zetta		10^{21}		Sextillion	1**z**	zepto		10^{-21}
1**E**	exa		10^{18}	Trillion	Quintillion	1**a**	atto		10^{-18}
1**P**	peta		10^{15}		Quadrillion	1**f**	femto		10^{-15}
1**T**	tera		10^{12}	Billion	Trillion	1**p**	pico		10^{-12}
1**G**	giga		10^{9}	Milliard	Billion	1**n**	nano		10^{-9}
1**M**	mega		10^{6}	Million	Million	1**u**	micro		10^{-6}

1**K**	kilo		10^3			1**m**	milli		10^{-3}
1**k**	kilo		10^3						
1**h**	hecto		10^2			1**c**	centi		10^{-2}
1**D, da**	deka		10^1			1**d**	deci		10^{-1}
1**R**			10^0						

Implicit Multiplication:		
3PI	9.424777960	3*PI
3pi	0+3e-12i	0+3pi
2sin(2PI/3)	1.732050807	2*sin(2*PI/3)
3(1+2)	9	3*(1+2)
(1+2)(1+2)(1+2)	27	(1+2)*(1+2)*(1+2)

1.1 Printf format

format specifiers have the following form

% [flags] [width] [.prec] [F|N|h|l|L] type_char

Each format specifier begins with the percent character (%).

After the % come the following optional specifiers, in this order:

Optional Format String Components

These are the general aspects of output formatting controlled by the optional characters, specifiers, and modifiers in the format string:

Component	**Optional/Required**	**What it Controls or Specifies**
[flags]	(Optional)	Flag character(s) Output justification, numeric signs, decimal points, trailing zeros, octal and hex prefixes
[width]	(Optional)	Width specifier Minimum number of characters to print, padding with blanks or zeros
[prec]	(Optional)	Precision specifier Maximum number of characters to print; for integers, minimum number of digits to print
[F|N|h|l|L]	(Optional)	Input size modifier Override default size of next input argument
type_char	(Required)	Conversion-type character

1.1.1 Format Specifier Conventions

Certain conventions accompany some of the printf format specifiers for the following conversions:

- %e or %E
- %f
- %g or %G
- %x or %X

Note: Infinite floating point numbers are printed as +INF and -INF. An IEEE not-a-number is printed as +NAN or -NAN.

1.1.1.1 %e or %E Conversions

The argument is converted to match the style

[-] d.ddd...e[+/-]ddd

where:

- one digit precedes the decimal point.
- the number of digits after the decimal point is equal to the precision.
- the exponent always contains at least two digits.

1.1.1.2 %f Conversions

The argument is converted to decimal notation in the style

[-] ddd.ddd...

where the number of digits after the decimal point is equal to the precision (if a non-zero precision was given).

1.1.1.3 %g or %G Conversions

The argument is printed in style **e**, **E** or **f**, with the precision specifying the number of significant digits.

Trailing zeros are removed from the result, and a decimal point appears only if necessary.

The argument is printed in style **e** or **f** (with some restraints) if **g** is the conversion character. Style **e** is used only if the exponent that results from the conversion is either greater than the precision or less than -4.

The argument is printed in style **E** if **G** is the conversion character.

1.1.1.4 %x or %X Conversions

For **x** conversions, the letters **a, b, c, d, e,** and **f** appear in the output.

For **X** conversions, the letters **A, B, C, D, E,** and **F** appear in the output.

1.1.2 Format flag

They can appear in any order and combination.

Flag	What it means
-	Left-justifies the result, pads on the right with blanks. If not given, it right justifies the result, pads on the left with zeros or blanks.
+	Signed conversion results always begin with a plus (+) or minus (-) sign.
blank	If value is nonnegative, the output begins with a blank instead of a plus; negative values still begin with a minus.
#	Specifies that arg is to be converted using an alternate form.

Note: Plus (+) takes precedence over blank () if both are given.

1.1.2.1 Alternate Forms for format Conversion

If you use the # flag conversion character, it has the following effect on the argument (arg) being converted:

Conversion character	How # affects the argument
c s d iu	No effect.
0	0 is prepended to a nonzero arg.
x X	0x (or 0X) is prepended to arg.
e E f	The result always contains a decimal point even if no digits follow the point. Normally, a decimal point appears in these results only if a digit follows it.
g G	Same as e and E, except that trailing zeros are not removed.

1.1.3 Format Width Specifiers

The width specifier sets the minimum field width for an output value.

Width is specified in one of two ways:

- directly, through a decimal digit string
- indirectly, through an asterisk (*)

If you use an asterisk for the width specifier, the next argument in the call (which must be an **int**) specifies the minimum output field width.

Nonexistent or small field widths do not cause truncation of a field. If the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

Width specifier	How output width is affected
n	At least *n* characters are printed. If the output value has less than *n* characters, the output is padded with blanks (right-padded if - flag given, left-padded otherwise).
0*n*	At least n characters are printed. If the output value has less than *n* characters, it is filled on the left with zeros.
*	The argument list supplies the width specifier, which must precede the actual argument being formatted.

1.1.4 Format Precision Specifiers

The printf precision specifiers set the maximum number of characters (or minimum number of integer digits) to print.

A printf precision specification always begins with a period (.) to separate it from any preceding width specifier.

Then, like [width], precision is specified in one of two ways:

- directly, through a decimal digit string
- indirectly, through an asterisk (*)

If you use an * for the precision specifier, the next argument in the call (treated as an int) specifies the precision.

If you use asterisks for the width or the precision, or for both, the width argument must immediately follow the specifiers, followed by the precision argument, then the argument for the data to be converted.

[.prec]	How Output Precision Is Affected
(none)	Precision set to default:

	= 1 for *d,i,o,u,x,X* types
	= 6 for *e,E,f* types
	= All significant digits for *g,G* types
	= Print to first null character for *s* types
	= No effect on *c* types
.0	For *d,i,o,u,x* types, precision set to default
	for *e,E,f* types, no decimal point is printed.
.n	*n* characters or *n* decimal places are printed.
	If the output value has more than n characters, the output might be truncated or rounded. (Whether this happens depends on the type character.)
.	The argument list supplies the precision specifier, which must precede the actual argument being formatted.

No numeric characters will be output for a field (i.e., the field will be blank) if the following conditions are all met:

- you specify an explicit precision of 0
- the format specifier for the field is one of the integer formats (d, i, o, u, or x)
- the value to be printed is 0

How [.prec] Affects Conversion

Char Type	Effect of [.prec] (.n) on Conversion
d	Specifies that at least n digits are printed.
i	If input argument has less than n digits,
o	output value is left-padded x with zeros.
u	If input argument has more than n digits,
x	the output value is not truncated.
X	
e	Specifies that n characters are
E	printed after the decimal point, and
f	the last digit printed is rounded.
g	Specifies that at most n significant
G	digits are printed.
c	Has no effect on the output.
s	Specifies that no more than n characters are printed.

1.1.5 Format Input-size Modifiers

These modifiers determine how printf functions interpret the next input argument, arg[f].

Prefix	Format specifier	Type specified
F	p s	A **far** pointer
N	n	A **near** pointer
h	d i o u x X	A **short int**
l	d i o u x X	A **long int**
l	e E f g G	A **double**
L	e E f g G	A **long double**
L	d i o u x X	An **__int64**
h	c C	A single-byte character
l	c C	A Wide character
h	s S	A single-byte character string

l	s S	A Wide character string

These modifiers affect how all the printf functions interpret the data type of the corresponding input argument arg.
Both *F* and *N* reinterpret the input variable *arg*. Normally, the *arg* for a *%p*, *%s*, or *%n* conversion is a pointer of the default size for the memory model.
h, l, and *L* override the default size of the numeric data input arguments. Neither *h* nor *l* affects pointer *(p,n)* types.

1.1.6 Format Conversion-Type Characters

The information in this table is based on the assumption that no flag characters, width specifiers, precision specifiers, or input-size modifiers were included in the format specifier.

Note: Certain conventions accompany some of these format specifiers.

Type Char	Expected Input	Format of output
Numerics		
d	Integer	**signed** decimal **integer**
i	Integer	**signed** decimal **integer**
o	Integer	**unsigned** octal **integer**
u	Integer	**unsigned** decimal **integer**
x	Integer	**unsigned** hexadecimal **int** (with **a, b, c, d, e, f**)
X	Integer	**unsigned** hexadecimal **int** (with **A, B, C, D, E, F**)
f	Floating point	**signed** value of the form [-]dddd.dddd.
e	Floating point	**signed** value of the form [-]d.dddd or e[+/-]ddd
g	Floating point	**signed** value in either e or f form, based on given value and precision. Trailing zeros and the decimal point are printed if necessary.
E	Floating point	Same as **e**; with **E** for exponent.
G	Floating point	Same as **g**; with **E** for exponent if e format used.
Characters		
c	Character	Single character
s	String pointer	Prints characters until a null-terminator is pressed or precision is reached
%	None	Prints the % character
Pointers		
n	Pointer to **int**	Stores (in the location pointed to by the input argument) a count of the chars written so far.
p	Pointer	Prints the input argument as a pointer; format depends on which memory model was used. It will be either XXXX:YYYY or YYYY (offset only).
Extra (application specific)		
b	Integer	%[width]b - binary (6 = 110)
C	Character	Special characters (6 = ACK '\006')
S	Floating point	Scientific (engineering) format (1000 = 1k)
B	Floating point	Computing (1024 = 1KB)
N	Floating point	Normalized (1000 = 1e3)
t	Integer	Date time (1000 = 16:m 40:s)
D	Floating point	Degrees (pi/3 = 59`59'59")
F	Floating point	Fractional "%F", pi -> 3+9/64, "%5F", pi -> 3+16/113

Infinite floating-point numbers are printed as +INF and -INF.

An IEEE not-a-number is printed as +NAN or -NAN.

1.2 Variables

View all defined variables and its values.

1.3 E series of preferred numbers

Index

- % -

- A -

- F -

- P -

Endnotes 2... (after index)

Back Cover