

Expression Calculator 2.26 Multithread Help

[About the Expression Calculator](#)
[Multithread Support / What's new / FAQ](#)
[User's guide for the Expression Calculator](#)
[Mathematical Repository](#)



(The Expression Calculator is dedicated to the genius of a swiss guy, Leonhard Euler)

(c) Daniel Doubrovkine
Stolen Technologies Inc. - University of Geneva
MCMXCVI - All Rights Reserved
e-mail welcome: dblock@infomaniak.ch / dubrov5@cui.unige.ch
<http://www.infomaniak.ch/~dblock/express.htm>

About the Expression Calculator

Expression Calculator is a powerful mathematical expressions evaluator. It is full 32 bit object oriented and is designed for Windows NT and Windows 95.

Expression Calculator 2.0 has been developed with (c) Borland Delphi 2.0 under Windows NT (TM).

Disclaimer: this software is free for personal use only. Any company or organization using the Expression Calculator should buy it from the author himself or from an authorized dealer. You may not modify, sell or abuse of this software without a total unforced consent of the author. You are encouraged by all means to distribute this software. Though the Expression Calculator has been diligently tested to provide absolutely correct evaluation results, I will still not be responsible for eventual errors caused directly or indirectly by the Expression Calculator itself, any kind of misuse, bugs in processors (looks familiar) or any external influence by any kind of life or artificial intelligence form. The author of this software will not be held responsible for any sort of consequences caused by a false calculation of the Expression Calculator, including space shuttle explosions and political crisis. (Okay, this is just a disclaimer, please don't be afraid to use this program...).

The Expression Calculator has not been ripped from anywhere, all material used has been listed below. The Expression Calculator includes uninherited TreeView32 (v.1.2) & MathCalc32 (v.2.1) (TThread) multithread objects (c) Daniel Doubrovkine (1996). Expression Calculator calculates a huge number of mathematical expressions with a satisfactory precision of between 10 and 20 decimals. Expression Calculator has a true multithread capability due to it's rigorous object oriented style and it's powerful calculation thread generator. This means you can calculate as many expressions as you like at the same time and launch any other calculation before a previous task is finished.

For bugs, problems, money, decent and serious proposals, please contact (best by e-mail):

Daniel Doubrovkine
11, chemin de la Clairière
1207 Geneva
Switzerland
tél & fax: 41 (0) 22 735 69 47
e-mail: dblock@infomaniak.ch / doubrov5@cui.unige.ch
Any kind of notes or support welcome!

The source of the multithread calculation object written in object pascal (Delphi 2.0) is available on the Expression Calculator's home page at <http://www.infomaniak.ch/~dblock/express.htm>. This source may be used freely for non lucrative purposes. For using it in shareware or small commercial software, please send US\$ 50.- to the author

Daniel Doubrovkine - Account No. C2-112.610 - Swiss Bank Corporation (Société de Banque Suisse)

For using it in large software projects, please contact the author. The price of the complete Expression Calculator source is US\$ 100.- and is available on demand. Of course you may download the source for free, but besides the appeal of your conscience, you should remember that Big Brother's watching you:)

Books and material used for writing this cool calculator: N. Wirth (Federal Politechnic Institute of Zürich...the guy who has written the bases of modern Pascal) - "Algorithms+Data Structures=Programs", R. Sedgewick (University of

Princeton) - "Algorithms in C language" (even though the Expression Calculator has been written in Object Pascal),
Bjarne Stroustrup (AT&T Bell Laboratories) - "The C++ Programming Language (Second Edition)", E.Hairer G.
Wanner (University of Geneva) - "Analysis by it's History" (gee, my math course, really excellent book... this is
where all the crazy [gamma](#) and [beta](#) functions come from), some other books i've checked as well as my algebra
course mostly by Mr. Haussmann (University of Geneva). There's also some code from the S.W.A.G Pascal support
team, which is doing a really great job.

Expression Calculator User's Guide

The general idea of the Expression Calculator is that you can write any valid mathematical expression and the program will calculate it for you. You must distinguish several parts of the Expression Calculator:

- parameters

when launching the Expression Calculator you can specify parameters at the command line:

small : initialize the Expression Calculator small
medium : initialize the Expression Calculator medium
scientific : initialize the Expression Calculator scientific
notitle : do not show the initialization bitmap

any other parameter will be ignored and the Expression Calculator will be initialized in the last mode it was started

- the Result Grid

The Result Grid provides a history for calculation. Clicking with the left button of the mouse on a cell will copy it's operational command to the input box for reevaluation. In order to insure maximum precision, the result is not taken, but will be reconsidered entirely. You can copy the corresponding result to the clipboard by clicking on the first column (containing the line numbers) with the left mouse button. Clicking with the right button on a running task will pop a menu that allows to abort, suspend and resume a task or all tasks. This menu contains functions for clearing the primes table as well as copying those primes to the clipboard. You may also reset the Expression Calculator, which is equivalent to pressing the **AC** button. The size of this grid is virtually infinite.

- the Input Box

The Input Box is used to enter an expression. You may enter any natural expressions the Expression Calculator understands. Check the repository for the complete list. The Expression Calculator is of course parenthesis and mathematical order aware. Try entering $2+2$ and pressing enter, the result should be somewhere around 4:)

- the Result Panel

The Result Panel shows the latest result of an operation. It contains a rounded to two decimals result as well as almost untruncated value. It also contains indicators for the calculation modes, Radian and Degree for the moment. The Result Panel has two arrows to wrap and unwrap the Expression Calculator from a scientific to a less scientific view.

- the Modes Panel

The Modes Panel allows to know at any moment the number of tasks running as well as to switch between the **RAD**ient and the **DEG**ree mode. The Mode Panel contains a button (right arrow) for rarely used but useful and powerful functions. By switching between the **DEG/RAD** (decimal modes), **HEX**adecimal, **BIN**ary and **OCT**al mode results and valid input will change consequently.

- the Functions' Panel

Divided in two parts, the Functions Panel contains operational flags for inverse, hyperbolic and negative functions:

ARC: inverse functions flag (**SIN** becomes **ARCSIN**)

HYP: hyperbolic functions flag (**SIN** becomes **SINH**, with the **ARC** enabled, **SIN** becomes **ARCSINH**)

NOT: inverse logical operations, **XOR** becomes **XNOR**, etc.

ABORT: prompts for aborting all running tasks

ABOUT: shows the about Expression Calculator Window

By pressing any other function button, the corresponding operation will be inserted at the cursor position of the Input Box. Check the repository for the complete list of functions.

- the Operations' Panel

Same as Functions Panel, this later contains the simple operations you may want to perform, as well as the variable support:

STO: store a single character variable (prompts a window with available variables)

RCL: restore a single character variable (prompts a window with available variables)

ANS: the last calculated expression

EXP: *10^{expression}

OFF: quit the expression calculator (you will be prompted if tasks running)

AC: clear all operations (you will be prompted if tasks running)

C: clear the current input

DEL: delete the actual input character

You can also resize the Expression Calculator. Shape will be kept using the registry and will be user private.

Mathematical Repository

This repository contains all the functions the Expression Calculator can perform with their mathematical description.

[+ \(plus\)](#)
[- \(minus\)](#)
[* \(product\)](#)
[/ \(division\)](#)
[% \(percentage\)](#)
[! \(integer division\)](#)
[^ \(power\)](#)
[! \(factor\)](#)
[\ \(root\)](#)
[abs](#)
[and](#)
[arccos](#)
[arccosh](#)
[arccot](#)
[arccoth](#)
[arccsc](#)
[arccsch](#)
[arcsec](#)
[arcsech](#)
[arcsin](#)
[arcsinh](#)
[arctan](#)
[arctanh](#)
[average](#)
[beta](#)
[binom](#)
[ceil](#)
[cos](#)
[cosh](#)
[cot](#)
[coth](#)
[csc](#)
[csch](#)
[e](#)
[exp](#)
[fib \(fibonacci\)](#)
[floor](#)
[frac](#)
[gamma \(Euler's Gamma\)](#)
[gcd \(greatest common divisor\)](#)
[genmers](#)
[int](#)
[lcm \(least common multiple\)](#)
[ln](#)
[log](#)
[logn](#)
[m \(modula\)](#)
[max](#)
[mersienne](#)

mersiennegen
mersgen
min
nand
new
nor
not
or
perfect
phi (eInd)
Pi, E
prime
primec
primen
random
round
sec
sech
shl
shr
sin
sinh
sqr
sqrt
tan
tanh
trunc
xnor
xor

+

description: simple addition

syntax: $x + y$

domain: $+ : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

example: $2 + 2 = 4$

-

description: simple subtraction

syntax: $x - y$

domain: $- : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

example: $2 - 8 = -6$

*

description: simple product

syntax: $x * y$

domain: $* : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

example: $2 * 2 = 4$

/

description: simple division
syntax: x / y
domain: $/ : (\mathbb{R} \times \mathbb{R}^*) \rightarrow \mathbb{R}^*$
example: $2 / -2 = -1$

m

description: modula division

syntax: $x \text{ m } y \mid x, y \text{ legal expressions, } y \neq 0$

domain: $m: (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{Z}$

example: $4 \text{ m } 2 = 0$

notes: errors will occur if combined to a variable like $pm2$ will be understood as a function and an error reported, you should put p in parenthesis to avoid this error, example: $(p)m2=0$ if $p=4$. Non integer values are truncated before the modula division.

%

description: left percentage of right expression

syntax: $x \% y$

domain: $\% : (R \times R) \rightarrow Z$

example: $4\%2=0.08$

notes: percentage is calculated sign independent, then multiplied by both signs.

⋮

description: integer division

syntax: $x \dot{|} y$

domain: $\dot{|} : (\mathbb{R} \times \mathbb{R}^*) \rightarrow \mathbb{R}$

example: $4.1 \dot{|} 2 = 2$

notes: first makes the division, then truncates the value

^

description: power, elevates a number into a power

syntax: $x ^ y$

domain: $^ : (R \times R) \rightarrow R$

example: $4 ^ 3 = 64$

!

description: factor

syntax: $x!$

domain: $! : \mathbb{R} \setminus \{ \mathbb{Z}^- \} \rightarrow \mathbb{R}$

example: $-2.1! = 9.71$

notes: uses Euler's [Gamma](#) function method for huge or non integer positive or negative numbers:

Step of 0.01 is used for the integral and infinity replaced by 100 which is definitely more than enough.

0! = 1 by definition

$$n! := \Gamma(n + 1)$$

\

description: root

syntax: $x \setminus y$

domain: $\setminus : \mathbb{R}^* \rightarrow \mathbb{R}$

and a pair root cannot be calculated for a negative number

example: $3 \setminus 8 = 2$

notes: $x \setminus y = y^{(1 / x)}$

XOR

description: bitwise xor

syntax: $x \text{ xor } y$

domain: $\text{xor}: (Z (\text{trunc}(R)) \times Z (\text{trunc}(R))) \rightarrow Z$

example: $2 \text{ xor } 4 = 6$

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is xored:

0	0	0
0	1	1
1	0	1
1	1	0

xnor

description: bitwise xnor

syntax: $x \text{ xnor } y$

domain: $\text{xnor}: (Z(\text{trunc}(R)) \times Z(\text{trunc}(R))) \rightarrow Z$

example: $2 \text{ xnor } 4 = -8$

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is xnored:

0	0	1
0	1	0
1	0	0
1	1	1

and

description: bitwise and

syntax: x and y

x & y

domain: and: (Z (trunc(R)) x Z (trunc(R))) -> Z

example: 3 and 9 = 1

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is anded:

0	0	0
0	1	0
1	0	0
1	1	1

nand

description: nand

syntax: $x \text{ nand } y$

domain: $\text{nand}: \mathbb{Z}(\text{trunc}(\mathbb{R})) \times \mathbb{Z}(\text{trunc}(\mathbb{R})) \rightarrow \mathbb{Z}$

example: $3 \text{ nand } 9 = 1$

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is nanded:

0	0	1
0	1	1
1	0	1
1	1	0

or

description: or

syntax: $x \text{ or } y$

domain: or: $(Z(\text{trunc}(R)) \times Z(\text{trunc}(R))) \rightarrow Z$

example: $2 \text{ or } 4 = 6$

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is ored:

0	0	0
0	1	1
1	0	1
1	1	1

nor

description: nor

syntax: $x \text{ nor } y$

domain: $\text{nor}: (\mathbb{Z}(\text{trunc}(\mathbb{R})) \times \mathbb{Z}(\text{trunc}(\mathbb{R}))) \rightarrow \mathbb{Z}$

example: $2 \text{ nor } 4 = -7$

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is nored:

0	0	1
0	1	0
1	0	0
1	1	0

Not

description: bitwise Not

syntax: Not (x)

domain: Not: Z (trunc(R)) $\rightarrow Z$

example: Not (1) = 2

notes: if the values of expressions for x and y are not integer, they are truncated
each bit is inversed:

0	1
1	0

Lcm

description: Least Common Multiple between up to as many arguments as wanted

syntax: Lcm (x , ... , y)

domain: Lcm: $(\mathbb{Z} \times \mathbb{Z}) \rightarrow \mathbb{Z}$

example: Lcm (14,4) =28

notes: Lcm (x,y):=(u [div gcd](#)(u,v))*v

ppcm does the same thing and stands for "le Plus Petit Commun Multiple"

Gcd

description: Greatest Common Divisor between up to as many arguments as wanted

syntax: Gcd (x , ... , y)

domain: Gcd: (Z x Z) -> Z

example: Gcd (3213,24) =3

notes: there's a very simple algorithm to find the greatest common divisor between two integers:

```
t: of x,y type;
while x <> 0) do begin
  t:=x-trunc(x/y)*y;      {t:=x mod y}
  x:=y;
  y:=t;
end;
gcd := x;
```

pgcd does the same thing and stands for "le Plus Grand Commun Diviseur"

Fib

description: Fibonacci integers

syntax: Fib(x)

domain: Fib : $\mathbb{N} \rightarrow \mathbb{N}$

example: Fib (56) = 2.258e11

notes: Fibonacci is defined this way:

Fib (x < 2) := x

Fib (x) := Fib(x-1) + Fib(x-2)

The current function uses a super fast iterative matrix method and is able to calculate Fibonacci numbers up to fib(2¹⁴).

Gamma

description: Euler's Gamma function (or simply Euler's function)

syntax: Gamma (x, step)

domain: Gamma : (R \ {integer negative numbers} x R) -> R

example: Gamma (0.5) = Sqrt (pi) = 1.77

notes:

$$\Gamma(a) := \int_0^{\infty} e^{-x} x^{a-1} dx$$

The function has been extended by Euler to negative values with the help of

$$\Gamma(a - 1) := \frac{\Gamma(a)}{a-1}$$

note that for negative values this is just a definition and the integral has here absolutely no sense

By the way, **Gamma (1/5) = Sqrt(Pi)** (which is calculated with the Gaussian integral, which is beyond the scope of this help file). Thus, this function diverges for integer negative values.

Check the [factorial](#) and Euler's [beta](#) functions for a concrete use of the gamma function.

shl

description: bitwise shift left

syntax: Shl (x , y)

domain: Shl : ($\mathbb{Z} \times \mathbb{Z}$) \rightarrow \mathbb{Z}

example: Shl (2,1) = 4

notes: shifts the values of x left of y positions, same as multiplying by 2^y

shr

description: bitwise shift right

syntax: $\text{Shr}(x, y)$

domain: $\text{Shr} : (\mathbb{Z} \times \mathbb{Z}) \rightarrow \mathbb{Z}$

example: $\text{Shr}(2,1) = 1$

notes: shifts the values of x right of y positions, same as dividing by 2^y

Min

description: chooses the smallest value between up to as many arguments as wanted

syntax: $\text{Min}(x, \dots, y)$

domain: $\text{Min} : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

example: $\text{Min}(1, 2) = 1$

Max

description: chooses the biggest value between up to as many arguments as wanted

syntax: $\text{Max}(x, \dots, y)$

domain: $\text{Max} : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$

example: $\text{Max}(1, 2, 234) = 234$

Phi Ind

description: Euler's indicator

syntax: $\Phi(x)$ or $e\text{Ind}(x)$

domain: $\Phi: \mathbb{Z} \rightarrow \mathbb{N}$

example: $\Phi(12) = 4$

notes: Euler's indicator is the number of prime numbers to the integer argument. $\Phi(x) = \Phi(-x)$.

$\Phi(x): \mathbb{Z}_m \rightarrow \mathbb{Z}_1 * \mathbb{Z}_2 \dots \mathbb{Z}_m$ is a isomorphism, meaning that the decomposition of any number into a product of prime numbers is unique. Euler has demonstrated this theorem and added that **$\Phi(x) = x * (1 - 1/p_1) * \dots * (1 - 1/p_n)$** , where $p_1 \dots p_n$ are prime number from the decomposition of x , taken once. Thus $12 = 2^2 * 3$ and $\Phi(12) = 12 * (1 - 1/2) * (1 - 1/3)$ which is always an integer value and is equal to 4. Thus, there are 4 primes inferior and to 12.

Frac

description: fractionary part of a number

syntax: $\text{Frac} (x)$

domain: $\text{Frac} : \mathbb{R} \rightarrow \mathbb{Z}$

example: $\text{Frac} (1.345) = 0.35$

notes: $\text{Frac} (x) = x - \text{Trunc} (x)$

Abs

description: absolute value of a number

syntax: $\text{Abs}(x)$

domain: $\text{Abs} : \mathbb{R} \rightarrow \mathbb{R}$

example: $\text{Abs}(-2) = 2$

Int

description: integer part of the number

syntax: $\text{Int}(x)$

domain: $\text{Int} : \mathbb{R} \rightarrow \mathbb{R}$

example: $\text{Int}(2.1) = 2$

Round

description: rounds a number to the closest integer

syntax: Round (x)

domain: Round : $\mathbb{R} \rightarrow \mathbb{Z}$

example: Round (-2.1) = -2

Trunc

description: truncates the number to an integer

syntax: $\text{Trunc}(x)$

domain: $\text{Trunc} : \mathbb{R} \rightarrow \mathbb{Z}$

example: $\text{Trunc}(2.1) = 2$

Log

description: 10 based logarithm of it's argument

syntax: $\text{Log} (x)$

domain: $\text{Log} : \mathbb{R}^+ \rightarrow \mathbb{R}$

example: $\text{Log} (10^{45}) = 45$

Logn

description: calculates any based logarithm

syntax: Logn (base, x)

domain: Logn : (R+, R+) -> R

example: Logn (2,8) = 3

Exp

description: exponential of the argument

syntax: $\text{Exp}(x)$

domain: $\text{Exp} : \mathbb{R} \rightarrow \mathbb{R}$

example: $\text{Exp}(3) = 20.09$

notes: raises E (2.72...) in the power of x

Ln

description: natural logarithm of the argument

syntax: $\text{Ln}(x)$

domain: $\text{Ln} : \mathbb{R}^+ \rightarrow \mathbb{R}$

example: $\text{Ln}(e) = 1$

Ceil

description: calculates the ceiling of the argument

syntax: $\text{Ceil}(x)$

domain: $\text{Ceil} : \mathbb{R} \rightarrow \mathbb{R}$

example: $\text{Ceil}(2.1) = 3$

$\text{Ceil}(-2.1) = -2$

Floor

description: calculates the floor of the argument

syntax: Floor (x)

domain: Floor : $\mathbb{R} \rightarrow \mathbb{R}$

example: Floor (2.1) = 2
Floor (-2.1) = -3

Sqrt

description: square root of an argument

syntax: Sqrt (x)

domain: Sqrt : $\mathbb{R} \setminus \{R-\}$ \rightarrow $\mathbb{R} \setminus \{R-\}$

example: Sqrt (4) = 2

Sqr

description: raises the argument to it's second power

syntax: $\text{Sqr}(x)$

domain: $\text{Sqr} : \mathbb{R} \rightarrow \mathbb{R}$

example: $\text{Sqr}(2) = 4$

Random

description: returns a random number \leq to the argument

syntax: `Random (x)`

domain: `Random : R+ -> R \ {R-}`

example: `Random (8) = 4`

New

description: new will prompt for redefining all the variables inside of the expression

syntax: New (expression)

notes: "new" alone will delete all the variables currently in memory

Sin

description: sine of the argument

syntax: Sin (x)

notes: check the calculator mode before you perform any trigonometric calculation
 $90^\circ = \pi / 2 = 100 \text{ grad}$

Cos

description: cosine of the argument

syntax: $\text{Cos} (x)$

notes: check the calculator mode before you perform any trigonometric calculation
 $90^\circ = \pi / 2 = 100 \text{ grad}$

Tan

description: tangent of the argument

syntax: Tan (x)

notes: check the calculator mode before you perform any trigonometric calculation
 $90^\circ = \pi / 2 = 100 \text{ grad}$

Cot

description: cotangent of the argument

syntax: $\text{Cotan} (x)$

notes: check the calculator mode before you perform any trigonometric calculation
 $90^\circ = \pi / 2 = 100 \text{ grad}$

Sinh

description: hyperbolic sine of the argument

syntax: Sinh (x)

Cosh

description: hyperbolic cosine of the argument

syntax: Cosh (x)

Tanh

description: hyperbolic tangent of the argument

syntax: $\text{Tanh}(x)$

Cotanh

description: hyperbolic cotangent of the argument

syntax: $\text{Cotanh}(x)$

Arcsin

description: inverse sine of the argument

syntax: $\text{Arcsinh} (x)$

notes: check the calculator mode before you perform any trigonometric calculation

Arccos

description: inverse cosine of the argument

syntax: $\text{Arccos}(x)$

notes: check the calculator mode before you perform any trigonometric calculation

Arctan

description: inverse tangent of the argument

syntax: $\text{Arctan}(x)$

notes: check the calculator mode before you perform any trigonometric calculation

Arccot

description: inverse cotangent of the argument

syntax: $\text{Arccot}(x)$

notes: check the calculator mode before you perform any trigonometric calculation

Arcsinh

description: inverse hyperbolic sine of the argument

syntax: $\text{Arcsinh}(x)$

Arccosh

description: inverse hyperbolic cosine of the argument

syntax: $\text{Arccosh}(x)$

Arctanh

description: inverse hyperbolic tangent of the argument

syntax: $\text{Arctanh}(x)$

Arccoth

description: inverse hyperbolic cotangent of the argument

syntax: $\text{Arcco}th(x)$

Sec

description: secant of the argument

syntax: $\text{Sec} (x)$

notes: $\text{Sec} (x) = 1 / \text{Cos} (x)$

Csc

description: cosecant of the argument

syntax: $\text{Csc} (x)$

notes: $\text{Csc} (x) = 1/\text{Sin} (x)$

Sech

description: hyperbolic secant of the argument

syntax: Sech (x)

notes: Sech (x) = 1/ Cosh (x)

Csch

description: hyperbolic cosecant of the argument

syntax: Csch (x)

notes: Csch (x) = 1/ Sinh (x)

Arcsec

description: inverse secant of the argument

syntax: Arcsec (x)

notes: Arcsec (x) = Arccos (1/ x)

Arccsc

description: inverse cosecant of the argument

syntax: $\text{Arccsc}(x)$

notes: $\text{Arccsc}(x) = \text{Arcsin}(1/x)$

Arcsech

description: inverse hyperbolic secant of the argument

syntax: Arcsech (x)

notes: Arcsech (x) = Arccosh (1/ x)

Arccsch

description: inverse hyperbolic cosecant of the argument

syntax: $\text{Arccsch}(x)$

notes: $\text{Arccsch}(x) = \text{Arcsinh}(1/x)$

Prime

description: latest prime number \leq argument

syntax: Prime (x)

domain: Prime : $\mathbb{R} \rightarrow \mathbb{N}$

example: Prime (86) = 83

notes: The routine of prime numbers calculation constructs a primes table while calculating higher prime numbers. It thus uses a powerful method to provide an immediate result if the prime number has already been calculated.
Use [primec](#) to get the number of primes inferior to a value.
Use [primen](#) to get the nth prime.

e

description: raises a number into a 10 power

syntax: $x e y$

domain: $e : \mathbb{R} \rightarrow \mathbb{R}$

example: $3e4 = 30000$

E, Pi

E = 2.71... $\ln(E)=1$ - Euler's number: $1+1+1/2!+1/3!+1/4!+\dots$

Pi = 3.14... - perimeter of half of the unit circle

Mersienne

description: finds the closest mersienne number inferior to the parameter (and $< 2^{32}$)

syntax: Mersienne (x)

domain: Mersienne : $[3, 2^{32}] \rightarrow (3, 7, 31, 127, 8191, 131071, 524287, 2147483647)$

example: Mersienne (145) = 127

notes: when $2^n - 1$ is prime it is said to be a Mersenne prime.

$$M(p) = 2^p - 1$$

$P(p) = 2^{(p-1)}(2^p - 1)$ is by the way a [perfect](#) number!

(taken from <http://www.utm.edu/research/primes>, a much bigger table of Mersienne, Prime and perfect numbers can be found there...)

MersienneGen

description: finds the closest [mersienne](#) generator inferior to the parameter (and < 32)

syntax: MersienneGen (x)

domain: Mersienne : [2,32] -> (2,3,5,7,13,17,19,31)

example: MersienneGen (8) = 7

MersGen

description: finds the [mersienne](#) number for a mersienne generator

syntax: MersGen (x)

domain: x must be a mersienne generator

GenMers

description: finds the [mersienne](#) generator for a mersienne number

syntax: GenMers (x)

domain: x must be a mersienne number

Perfect

description: finds the closest inferior perfect number to the parameter

syntax: Perfect (x)

domain: Perfect : $\mathbb{R}^+ \rightarrow ()$

example: Perfect (1231) = 496

notes: a positive integer n is called a **perfect number** if it is equal to the sum of all of its positive divisors, excluding n itself.

$$6, 6=3*2*1=3+2+1$$

$$28, 28=14*7*4*2*1=14+7+4+2+1$$

there is a direct relation to the **mersienne** primes $M(n) := (2^{n-1})$:

- k is an even perfect number if and only if it has the form $2^{(n-1)}*(2^n-1)$.

- if 2^n-1 is prime, then so is n.

Finally, it is not known whether or not there is an odd perfect number, but if there is one it is big!

(taken from <http://www.utm.edu/research/primes>)

Multithread Expression Calculator - what's new / frequently asked questions?

What's new?

version 2.0 of the Expression Calculator has been rewritten using a powerful native multithread support of 32 bit systems such as Windows NT and ... well, still a `_multithread_` support ... Windows 95.

the version 2.01 has a faster prime generation algorithm, speeding up 2.5 time approx the previous versions' calculations and eating 3 times less memory

(version 2.02)

- the following functions will now accept up to 255 parameters: [max](#), [min](#), [gcd](#), [lcm](#)
- [max](#), - [min](#), - [gcd](#), - [lcm](#), - [gamma](#) & - [logn](#) will now work correctly as their `_positive_` versions
- added [average](#) function

(version 2.03/2.04/2.05/2.06)

- faster calculation interrupt routine using native Win NT / 95 thread support instead of pseudo single task interrupt requests (you may now interrupt more than one calculation thread at the same time...you'll really see the difference in the Expression Calculator response with a slow machine)
- added suspend and resume threads routine (right button pops up the menu on the result grid)
- added command line parameters for startup (small, medium)
- added system information at the initialization and to the about box, providing processor(s), operating system, computer name and memory information
- added the help speed button
- added sleep delay at multithread primes generator at second instance, speeds up main primes generator thread dramatically as prime threads are added

(version 2.1)

- corrected a severe bug in the prime generator
- corrected a bug in functions accepting more than 2 parameters, up to as many arguments as wanted can be specified for [average](#), [max](#), [min](#), [gcd](#) & [lcm](#).
- added `primeC`, `primeN`
- added Euler's beta function
- added [Binomial](#) function

(version 2.11/2.12)

- added three functions (accessible from the menu of the calculation results grid by pressing the right mouse button), (special section for first two):
 - clear the primes' table, which resets all primes calculations that have been performed and frees all used memory
 - copy the calculated primes to clipboard, copies the calculated primes to clipboard and delimits them by commas; this function is pretty slow and is a task that you cannot yet cancel
 - reset the expression calculator
- corrected the pushbutton bug with the Arc and the Not buttons which could not be both down at the same time
- primes are cleared when the AC button is pressed or the Reset item in the grid menu chosen

- the result grid returns truncated values (with non-truncated values if different)
- the Expression Calculator will start in the last used mode (scientific / small / medium). Registry is used to store this information and it is user private.

(version 2.2)

- Expression Calculator can now be resized freely, buttons and panels follow during resizing, shape is kept at every restart. Registry is used to store this information and it is user private.
- The Expression Calculator will be set to the position and form of a previous session unless "small", "medium" or "scientific" parameter specified

(version 2.21)

- corrected last line highlight (when the last line in the grid is selected, the text copied to the entry box is also highlighted)
- last operation selection has changed, it will scroll through the calculations performed (DOSKEY style) instead of simply copying the last operation a thousand times
- standard Windows copy (Ctrl-C), cut (Ctrl-X & Shift-Del) and paste (Ctrl-V & Shift Ins) functions do work in the edit box
- untruncated possible results are bigger (up to $\sim 2^{120}$ is shown in normal mode instead of the $X \cdot 10^{\text{style}}$)
- when you press the first column (column with line numbers) on the grid, the result (not the expression) is copied to the clipboard
- you can now minimize the Expression Calculator

(version 2.22/2.23/2.24/2.25)

- corrected a dramatic lack of precision in the gamma, beta and factor functions for values over 70, the possible factorial, gamma and beta limits have been taken down to 1508!, but the results are precise for all of the values
- corrected value output for the $X \cdot 10^{\text{style}}$ for huge numbers
- you may start a fractional number with a point directly, 0.14322 may be written .14322
- corrected precision for very small negative values (47886.5-3323.4-6136.6-5547.6-14783.3-6958.8-1038.4-10098.4 did not give exactly 0)
- corrected precise result for negative values
- the initialization screen and some other bitmaps are full 256 color when using 256 color screen mode
- hexadecimal and octal modes work for negative numbers
- new parameter added to the Expression Calculator launch: "notitle" will not show the initialization bitmap when loading the Expression Calculator
- corrected the eInD (Phi) function, it will not affect the prime function any longer and will work correctly
- Expression Calculator will never start out of screen bounds any longer
- Init screen moved to front as the Expression Calculator initializes

(version 2.26)

- added code to avoid multithread concurrence bug under Windows NT 4.0, which caused the creation of unsuspendable and uninterruptable tasks during processor overload
- clicking on a pushed hex, bin, oct or rad button will reset the deg mode

FAQ (frequently asked questions):

- Why does anybody need a multithread calculator?

Hmmm... i am not answering political questions. (Yep, the answer is as stupid as the question).

- Does multithreading make the calculation more precise?

No it doesn't. (and check question 1).

- Is the source available?

Yes it is. The source of the multithread calculation object written in object pascal (Delphi 2.0) is available on the Expression Calculator's home page at <http://www.infomaniak.ch/~dblock/express.htm>. This source may be used freely for non lucrative purposes. For using it in shareware or small commercial software, please send US\$ 50.- to the author

Daniel Doubrovkine - Account No. C2-112.610 - Swiss Bank Corporation (Société de Banque Suisse)

For using it in large software projects, please contact the author. The price of the complete Expression Calculator source is US\$ 100.- and is available on demand. Of course you may download the source for free, but besides the appeal of your conscience, you should remember that Big Brother's watching you:)

For curious individuals:

What is an object?

An object type is a data structure that contains a fixed number of components. Each component is either a field (which contains data of a particular type), a method, which performs an operation on the object; or a property. An object type can inherit components from another object type. The inheriting object is a descendant and the object inherited from is an ancestor. More than one instance of an object is possible, thus each time a calculation was performed with the Expressions Calculator 1.x, a new calculation object was created, the calculation executed inside the object and the result output by the object itself as the job was finished. The object is then cleared by the EXEcute routine that has created it.

There's a Microsoft Win32S definition of the object, which is a unit of information that resides in a compound document and whose behavior is constant no matter where it is located or used. Another one from the C++ reference manual: an object is a region of storage.

What is a class?

A class is a user defined type. A class contains objects, thus a class is a more complete object. A class contains functions for it's initialization, manipulation and clean-up.

What is a thread?

A thread is a very particular class. It is a separate task under a multitasking environment that supports it (Windows NT for example). Native multithreading consist of giving an adequate time to every thread running depending on the thread's priority. Unlike under Windows 3.x which gave the hand to an application and waited until the later gave it back, Windows NT, OS/2, Unix systems, and in some way Windows 95, destribute this time and take back the hand whenever the operating system wants it. The Expression Calculator takes complete advantage of this technique! This allows an optimal use of the CPU.

Okay, what about the Expression Calculator?

Without the threads the Expression Calculator was always waiting for the object to finish the calculation. By executing another operation, before the previous one finished, the Expression Calculator was reentering the same execution routine again, thus dramatically slowing the calculations running and eating lot's of memory for nothing. Thus it was practically impossible to cancel an operation on the bottom of this execution (the first run), since it was running at probably 5% of it's normal speed and came back to it's normal state only after all the tasks on top were finished.

Multithreading solves the problem completely. Every time the user asks for a new calculation to be done, the Expression Calculator creates a new calculation thread instance giving it the necessary parameters such as the input string and the calculation mode, as well as the target for the output result. The execution routine which generated the thread terminates before the thread has actually even started to work. The thread exists separately and is protected by the operating system. It performs the calculation at it's maximum speed affecting very little the GUI performance, but using the CPU at 100%. The thread itself will output the result as it is finished and will destroy itself after all jobs have been done. Threads communicate with the Expression Calculator through private pipes, without affecting the performance in any way. You must still know that every thread initialized slows down all the others already running. Each thread will receive a similar portion of time from the CPU. There's one exception, while generating prime numbers with requests from more than one thread, there is only the working thread that is taking CPU time, those waiting for the result to complete loop eating very little CPU time by "sleeping" a certain ammount of CPU cycles, giving maximum time possible to the working task. Virtually an infinite number of threads can be created and any of the threads very easilly cancelled.

What part of the Expression Calculator is a true thread?

Every window (form) under Windows NT/95 is a thread. The calculation thread generator (when you press the EXE button) is a thread itself, so you never wait for it to finish. The calculation routine is a fully independent 32 bit thread.

And finally. what is really the speed difference?

8 times faster for prime numbers calculation with the same algorithm for example... Any arguments against multithreading?

Average

description: finds the average number between up to as many arguments as wanted

syntax: Average (x,...,y)

domain: Average: $\mathbb{R} \rightarrow \mathbb{R}$

example: Average(1,2) = 1.5

Beta

description: computes Euler's Beta function

syntax: Beta(a,b,dt)

notes: check the [gamma](#) function for more details:

$$B(a, \beta) = \frac{\Gamma(a)\Gamma(\beta)}{\Gamma(a + \beta)} = \int_0^1 (1-t)^{a-1} t^{\beta-1} dt$$

The Beta function is a direct definition from the calculation of the product of two [Gamma](#) functions which is itself a double integral, note that for negative values the integral has no sense and Beta of negative values is a pure definition from the [Gamma](#) function.

PrimeC

description: returns the number of primes inferior to it's argument

syntax: PrimeC(x)

domain: $\mathbb{R} \setminus \{0\} \rightarrow \mathbb{N}$

notes: 1 is considered as a prime and the argument itself is included in the count if it is a prime number. Check the [prime](#), [primen](#) and [phi](#) functions. Check the [prime](#) and the [primen](#) functions.

PrimeN

description: returns the nth prime

syntax: PrimeN(n)

domain: $\mathbb{N} \rightarrow \mathbb{N}$

notes: 1 is considered as a prime, so `primen(1)` will return 1 and not 2! Check the [prime](#), [primec](#) and [phi](#) functions.

Binom

description: binomial coefficients
syntax: Binom(n,j)
domain: $\mathbb{R} \rightarrow \mathbb{R}$ (with restrictions of [Gamma](#) function)
notes: Binom(n,j):= $n! / (j! (n-j)!)$

