Expression Calculator 2.1 Multithread Help

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

Stolen Technologies Inc. - University of Geneva

e-mail welcome: dblock@infomaniak.ch / doubrov5@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 developped with (c) Borland Delphi 2.0 under Windows NT (TM).

Disclaimer: this software is absolutely free for any kind of use. You may not modify, sell or abuse of it without a total unforced conscent of the author. You are encouraged by all means to distribute this software. Though the Expression Calculator has been diligently tested to provide absoulutely 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 meterial 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 Calcualtor 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 rigourous object oriented style and it's powerfull 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!

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), E.Hairer G. Wanner (University of Geneva) - "Analysis by it's History" (gee, my math course, really excellent book), some other books i've checked as well as my algebra course. There's also some code from the S.W.A.G Pascal support team.

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
  any other parameter will be ignored and the Expression Calculator will be initialized in the scientific mode

- 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. 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. The size of this grid is virtually infinite.

- the Input Box

  The Input Box is used to enter an expression. You max 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 almost untruncated value. It also contains indicators for the calculation modes, Radient 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

Mathematical Repository

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

+

description: simple addition
syntax: x + y
domain: + : (R x R) -> R
example: 2 + 2 = 4

-

description: simple substraction
syntax: x - y
domain: - : (R x R) -> R
example: 2 - 8 = -6

*

description:	simple product
syntax:	x * y
domain:	* : (R x R) -> R
example:	2 * 2 = 4

/

description: simple division
syntax: x / y
domain: / : (R x R*) -> R*
example: 2 / -2 = -1

m

description: modula division
syntax: x m y | x,y legal expressions, y <> 0
domain: m: (R x R) -> Z
example: 4 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 x R) -> Z
example: 4%2=0.08
notes: percentage is calculated sign independent, then multiplied by both signs.

¦

description: integer division
syntax: x ¦ y
domain: ¦ : (R x R*) -> R
example: 4.1¦ 2 = 2
notes: first makes the division, then truncates the value

^

description: power, elevates a number into a power
syntax: x ^ y
domain: ^ : (R x R) -> R
example: 4 ^ 3 = 64

!

description: factor
syntax: x !
domain: ! : R \ { Z- } -> 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 definetely more than enough.
0! = 1 by definition

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

\

description: root
syntax: x \ y
domain: \ : R* -> R
and a pair root cannot be calculated for a negative number
example: 3 \ 8 = 2
notes: x \ y = y^(1 / x)

xor

description:	bitwise xor
syntax:	x xor y
domain:	xor: (Z (trunc(R)) x Z (trunc(R))) -> Z
example:	2 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 xnor y
domain: xnor: (Z (trunc(R)) x Z (trunc(R))) -> Z
example: 2 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 nand y
domain:	nand: Z (trunc(R)) x Z (trunc(R))) -> Z
example:	3 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 or y
domain: or: (Z (trunc(R)) x Z (trunc(R))) -> Z
example: 2 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 nor y
domain:	nor: (Z (trunc(R)) x Z (trunc(R))) -> Z
example:	2 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)) -> 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: (Z x Z) -> 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 : N -> 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(2^14).

Gamma

description: Euler's Gamma functions
syntax: Gamma (x, step)
domain: Gamma : (R \ {integer negative numbers} x R) -> R
example: Gamma (0.5) = Sqrt (pi) = 1.77
notes:

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

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

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

.

By the way, **Gamma (1/5) = Sqrt(Pi)**. 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 : (Z x Z) -> 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: Shr (x , y)
domain: Shr : (Z x Z) -> Z
example: 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: Min (x , ... , y)
domain: Min : (R x R) -> R
example: Min (1,2) = 1

Max

description: chooses the biggest value between up to as many arguments as wanted
syntax: Max (x , ... , y)
domain: Max : (R x R) -> R
example: Max (1,2,234) = 234

Phi
Ind

description: Euler's indicator
syntax: Phi (x) or eInd (x)
domain: Phi : Z -> 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): Zm -> Z1*Z2...Zm is a isomorphism, meaning that the decomposition of any number into a product of prime numbers is unique. Euler has demonstrated this theoreme and added that **Phi (x) = x * (1-1/p1)*...*(1-1/pn)**, where p1...pn 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 inferiour and to 12.

Frac

description:	fractionary part of a number
syntax:	Frac (x)
domain:	Frac : R -> Z
example:	Frac (1.345) = 0.35
notes:	Frac (x) = x - Trunc (x)

Abs

description:	absolute value of a number
syntax:	Abs (x)
domain:	Abs : R -> R
example:	Abs (-2) = 2

Int

description: integer part of the number
syntax: Int (x)
domain: Int : R -> R
example: Int (2.1) = 2

Round

description: rounds a number to the closest integer
syntax: Round (x)
domain: Round : R -> Z
example: Round (-2.1) = -2

Trunc

description:	truncates the number to an integer
syntax:	Trunc (x)
domain:	Trunc : R -> Z
example:	Trunc (2.1) = 2

Log

description: 10 based logarithm of it's argument
syntax: Log (x)
domain: Log : R+ -> R
example: 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:	Exp (x)
domain:	Exp : R -> R
example:	Exp (3) = 20.09
notes:	raises E (2.72...) in the power of x

Ln

description: natural logorithm of the argument
syntax: Ln (x)
domain: Ln : R+ -> R
example: Ln (E) = 1

Ceil

description: calculates the ceiling of the argument
syntax: Ceil (x)
domain: Ceil : R -> R
example: Ceil (2.1) = 3
Ceil (-2.1) = -2

Floor

description: calculates the floor of the argument
syntax: Floor (x)
domain: Floor : R -> R
example: Floor (2.1) = 2
Floor (-2.1) = -3

Sqrt

description: square root of an argument
syntax: Sqrt (x)
domain: Sqrt : R \ {R-} -> R \ {R-}
example: Sqrt (4) = 2

Sqr

description: raises the argument to it's second power
syntax: Sqr (x)
domain: Sqr : R -> R
example: Sqr (2) = 4

Random

description: returns a random number <= 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 trignonometric calculation
90° = pi / 2 = 100 grad

Cos

description: cosine of the argument
syntax: Cos (x)
notes: check the calculator mode before you perform any trignonometric calculation
90° = pi / 2 = 100 grad

Tan

description: tangent of the argument
syntax: Tan (x)
notes: check the calculator mode before you perform any trignonometric calculation
90° = pi / 2 = 100 grad

Cot

description: cotangent of the argument
syntax: Cotan (x)
notes: check the calculator mode before you perform any trignonometric calculation
90° = pi / 2 = 100 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: Tanh (x)

Cotanh

description:	hyperbolic cotangent of the argument
syntax:	Cotanh (x)

Arcsin

description: inverse sine of the argument
syntax: Arcsinh (x)
notes: check the calculator mode before you perform any trignonometric calculation

Arccos

description: inverse cosine of the argument
syntax: Arccos (x)
notes: check the calculator mode before you perform any trignonometric calculation

Arctan

description: inverse tangent of the argument
syntax: Arctan (x)
notes: check the calculator mode before you perform any trignonometric calculation

Arccot

description: inverse cotangent of the argument
syntax: Arccot (x)
notes: check the calculator mode before you perform any trignonometric calculation

Arcsinh

description: inverse hyperbolic sine of the argument
syntax: Arcsinh (x)

Arccosh

description: inverse hyperbolic cosine of the argument
syntax: Arccosh (x)

Arctanh

description: inverse hyperbolic tangent of the argument
syntax: Arctanh (x)

Arccoth

description: inverse hyperbolic cotangent of the argument
syntax: Arccoth (x)

Sec

description: secant of the argument
syntax: Sec (x)
notes: Sec (x) = 1/ Cos (x)

Csc

description: cosecant of the argument
syntax: Csc (x)
notes: Csc (x) = 1/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: Arccsc (x)
notes: Arccsc (x) = 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: Arccsch (x)
notes: Arccsch (x) = Arcsinh (1/ x)

Prime

description: latest prime number <= argument
syntax: Prime (x)
domain: Prime : R -> N
example: Prime (86) = 83
notes: The routine of prime numbers calculation constructs a primes table while calculating higer prime numbers. It thus uses a powerful method to provide and immediate result if the prime number has already been calculated.
Use primec to get the number of primes inferiour 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 : R -> R
example:	3e4 = 30000

E, Pi

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

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] -> (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 : R+ -> ()
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 mutithread 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

For curious individuals (FAQ):

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.

What is a thread?

A thread is a very particular object. 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: R -> 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:

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

PrimeC

description: returns the number of primes inferiour to it's argument
syntax: Primec(x)
domain: R\{R-}   -> 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: N -> 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:	R -> R (with restrictions of Gamma function)
notes:	Binom(n,j):=n! / (j! (n-j)!)