NAME

mpexpr − Evaluate an expression with multiple precision math

SYNOPSIS

package require Mpexpr
mpexpr *arg* ?*arg arg ...*?
mpformat *formatString* ?*arg arg ...*?
global mp_precision

DESCRIPTION

Mpexpr is based on Tcl's native *expr* command, and shares many similarities with *expr*. Mpexpr performs all of its calculations using an arbitrary precision math package.

Mpexpr concatenates *arg*'s (adding separator spaces between them), evaluates the result as a Tcl expression, and returns the value. The operators permitted in Tcl expressions are a subset of the operators permitted in C expressions, and they have the same meaning and precedence as the corresponding C operators. Expressions almost always yield numeric results (integer or floating-point values). For example, the expression

mpexpr 8.2 + 6

evaluates to 14.2. Tcl expressions differ from C expressions in the way that operands are specified. Also, Tcl expressions support non-numeric operands and string comparisons.

OPERANDS

A Tcl expression consists of a combination of operands, operators, and parentheses. White space may be used between the operands and operators and parentheses; it is ignored by the expression processor. Where possible, operands are interpreted as integer values. Integer values may be specified in decimal (the normal case), in octal (if the first character of the operand is **0**), or in hexadecimal (if the first two characters of the operand are **0x**). If an operand does not have one of the integer formats given above, then it is treated as a floating-point number if that is possible. Floating-point numbers may be specified in any of the ways accepted by an ANSI-compliant C compiler (except that the "f", "F", "l", and "L" suffixes will not be permitted in most installations). For example, all of the following are valid floating-point numbers: 2.1, 3., 6e4, 7.91e+16. If no numeric interpretation is possible, then an operand is left as a string (and only a limited set of operators may be applied to it).

Operands may be specified in any of the following ways:

[1] As an numeric value, either integer or floating-point.

[2] As a Tcl variable, using standard **$** notation. The variable's value will be used as the operand.

[3] As a string enclosed in double-quotes. The expression parser will perform backslash, variable, and command substitutions on the information between the quotes, and use the resulting value as the operand

[4] As a string enclosed in braces. The characters between the open brace and matching close brace will be used as the operand without any substitutions.

[5] As a Tcl command enclosed in brackets. The command will be executed and its result will be used as the operand.

[6] As a mathematical function whose arguments have any of the above forms for operands, such as "**sin($x)**". See below for a list of defined functions.

Where substitutions occur above (e.g. inside quoted strings), they are performed by the expression processor. However, an additional layer of substitution may already have been performed by the command parser before the expression processor was called. As discussed below, it is usually best to enclose expressions in braces to prevent the command parser from performing substitutions on the contents.

For some examples of simple expressions, suppose the variable **a** has the value 3 and the variable **b** has the value 6. Then the command on the left side of each of the lines below will produce the value on the following line:

mpexpr 3.1 + $a
6.1

mpexpr 2 + "$a.$b"
5.6

mpexpr 4*[llength "6 2"]
8

mpexpr {{word one} < "word $a"}
0

OPERATORS

The valid operators are listed below, grouped in decreasing order of precedence:

− + ~ !
Unary minus, unary plus, bit-wise NOT, logical NOT. None of these operands may be applied to string operands, and bit-wise NOT may be applied only to integers.

*** / %**
Multiply, divide, remainder. None of these operands may be applied to string operands, and remainder may be applied only to integers. The remainder will always have the same sign as the divisor and an absolute value smaller than the divisor.

+ −
Add and subtract. Valid for any numeric operands.

<< >>
Left and right shift. Valid for integer operands only. Integers in mpexpr are not limited to a machine word and do not use two's complement format. Therefore shifting will not include a sign bit.

< > <= >=
Boolean less, greater, less than or equal, and greater than or equal. Each operator produces 1 if the condition is true, 0 otherwise. These operators may be applied to strings as well as numeric operands, in which case string comparison is used.

== !=
Boolean equal and not equal. Each operator produces a zero/one result. Valid for all operand types.

&
Bit-wise AND. Valid for integer operands only.

^
Bit-wise exclusive OR. Valid for integer operands only.

|
Bit-wise OR. Valid for integer operands only.

&&
Logical AND. Produces a 1 result if both operands are non-zero, 0 otherwise. Valid for numeric operands only (integers or floating-point).

||
Logical OR. Produces a 0 result if both operands are zero, 1 otherwise. Valid for numeric operands only (integers or floating-point).

*x***?***y***:***z*
If-then-else, as in C. If *x* evaluates to non-zero, then the result is the value of *y*. Otherwise the result is the value of *z*. The *x* operand must have a numeric value.

See the C manual for more details on the results produced by each operator. All of the binary operators group left-to-right within the same precedence level. For example, the command

mpexpr 4*2 < 7

returns 0.

The **&&**, **||**, and **?:** operators have "lazy evaluation", just as in C, which means that operands are not evaluated if they are not needed to determine the outcome. For example, in the command

mpexpr {$v ? [a] : [b]}

only one of **[a]** or **[b]** will actually be evaluated, depending on the value of **$v**. Note, however, that this is only true if the entire expression is enclosed in braces; otherwise the Tcl parser will evaluate both **[a]** and **[b]** before invoking the **expr** command.

MATH FUNCTIONS

Mpexpr supports the following mathematical functions in expressions. *x* and *y* are integer or floating point values; *i*, *j* and *c* are integer values;

Math functions compatible with *expr*:

acos(*x*) Arc cosine of *x*.

asin(*x*) Arc sine of *x*.

atan(*x*) Arc tangent of *x*.

atan2(*x,y*) Arc tangent of *x* / *y*.

ceil(*x*) Least integral value greater than or equal to *x*.

cos(*x*) Cosine of *x*.

cosh(*x*) Hyperbolic cosine of *x*.

exp(*x*) Exponential function e ** *x*.

floor(*x*) Greatest integral value less than or equal to *x*.

fmod(*x,y*) Remainder of *x* divided by *y*.

hypot(*x,y*) Euclidean distance of sqrt(*x* * *x* + *y* * *y*).

log(*x*) Natural logarithm of *x*.

log10(*x*) Base-10 logarithm of *x*.

pow(*x,y*) *x* raised to the *y* power.

sin(*x*) Sine of *x*.

sinh(*x*) Hyperbolic sine of *x*.

sqrt(*x*) Square root of *x*.

tan(*x*) Tangent of *x*.

tanh(*x*) Hyperbolic tangent of *x*.

abs(*x*) Returns the absolute value of *x*. *x* may be either integer or floating-point, and the result is returned in the same form.

double(*x*) If *x* is a floating value, returns *x*, otherwise converts *x* to floating and returns the converted value.

int(*x*) If *x* is an integer value, returns *x*, otherwise converts *x* to integer by truncation and returns the converted value.

round(*x*) If *x* is an integer value, returns *x*, otherwise converts *x* to integer by rounding and returns the converted value.

Additional *mpexpr* functions:

root(*x,y*) The *yth* root of *x*.

frem(*x,y*) Remove all occurance of factor*y* from number *x*.

minv(*x,y*)	Inverse of *x* modulo *y*.
gcd(*x,y*)	Greatest common divisor of *x* and *y*.
lcm(*x,y*)	Least common multiple of *x* and *y*.
max(*x,y*)	Maximum of *x* and *y*.
min(*x,y*)	Minimum of *x* and *y*.
pi()	Value of pi.
fib(*i*)	Fibonacci number of integer *i*.
fact(*i*)	Factorial of integer *i*.
pfact(*i*)	Product of prime numbers up to integer *i*.
lfactor(*i,c*)	Lowest prime factor of integer *i*, trying count *c* times.
iroot(*i,j*)	Integer root *j* of integer *i*.
gcdrem(*i,j*)	Relatively prime of greatest common divisior of *i* divided by *j*.
perm(*i,j*)	Permutations of *i* taking *j* at a time: *i* ! / (*i* - *j*) !.
comb(*i,j*)	Combinations of *i* taking *j* at a time: *i* ! / (*j* ! * (*i* - *j*) !) .
prime(*i,c*)	Return 0 if *i* is not prime, return 1 if *i* probably is prime. Test for primality count *c* times. The chance of a non-prime passing this test is less than (1/4)^count. For example, a count of 100 fails for only 1 in 10^60 numbers.
relprime(*i,j*)	Return 1 if *i* and *j* are relatively prime to each other, 0 otherwise.

TYPES, OVERFLOW, AND PRECISION

Computations are performed using arbitrary fixed and floating point values. Native machine values (*int, long, IEEE 754 floating point, etc.*) and instructions are not used. Conversion among internal representations for integer, floating-point, and string operands is done automatically as needed. For arithmetic computations, integers are used until some floating-point number is introduced, after which floating-point is used. For example,

mpexpr 5 / 4

returns 1, while

mpexpr 5 / 4.0
mpexpr 5 / ([string length "abcd"] + 0.0)

both return 1.25. Floating-point values are always returned with a ‘‘.’’ or an ‘‘e’’ so that they will not look like integer values. For example,

mpexpr 20.0/5.0

returns ‘‘4.0’’, not ‘‘4’’.

The global variable **mp_precision** determines the number of significant digits that are retained during evaluation. If **mp_precision** is unset then 17 digits of precision are used. The maximum value of **mp_precision** is 10000. Note that larger values for **mp_precision** will require increasingly longer execution times. Setting **mp_precision** to an illegal value will generate an error.

STRING OPERATIONS

String values may be used as operands of the comparison operators, although the expression evaluator tries to do comparisons as integer or floating-point when it can. If one of the operands of a comparison is a string and the other has a numeric value, the numeric operand is converted back to a string using the C *sprintf* format specifier **%d** for integers and **%g** for floating-point values. For example, the commands

mpexpr {"0x03" > "2"} mpexpr {"0y" < "0x12"}

both return 1. The first comparison is done using integer comparison, and the second is done using string comparison after the second operand is converted to the string ‘‘18’’. Because of Tcl’s tendency to treat values as numbers whenever possible, it isn’t generally a good idea to use operators like **==** when you really want string comparison and the values of the operands could be arbitrary; it’s better in these cases to use the **string compare** command instead.

mpformat formats a string in the style of Tcl’s native *format* command. Mpformat will interpret numeric arguments as arbitrary precision numbers. Mpformat performs limited % substitution on the output string. The following may be specified:

% [-] [width[.precision]] formatChar

- Specifies left justification; right justification is the default.

width.precision
Specifies optional width and precision. Default precision is 8. Width and/or precision may be specified as *****, in which the next argument will be used for the width or precision value.

Format character and result

d Format next argument as integer, truncating after the decimal point.

f Format next argument in decimal floating point.

e Format next argument in scientific notation.

r, **R** Format next argument as rational fraction x / y.

N Format next argument as numerator only of rational fraction x / y.

D Format next argument as denominator only of rational fraction x / y.

o Format next argument in octal format, with leading ’0’; floating point argument formatted as octal rational fraction x / y.

x Format next argument in hexadecimal format, with leading ’0x’; floating point formatted argument as hexadecimal rational fraction x / y.

b Format next argument in binary format, with leading ’0b’; floating point argument formatted as binary rational fraction x / y.

s Format next argument as string.

c Format next argument as single character value.

% Format single literal %.

Other characters in format string

\n Format ASCII newline.

\r Format ASCII carriage return.

\t Format ASCII tab.

\f Format ASCII form feed.

\v Format ASCII vertical tab.

\b Format ASCII backspace.

NOTES

Mpexpr is based on Tcl 7.6 'tclExpr.c' and David Bell's 'Calc' program. This man page is largely borrowed from Tcl 7.6 as well, as is the mpexpr test suite.

See the files README and INSTALL for additional information.

AUTHOR

Tom Poindexter, tpoindex@nyx.net, Talus Technologies, Inc., Highlands Ranch, CO. http://www.nyx.net/˜tpoindex

Version 1.0 released November, 1998.