Documentation - Simple mathematical interpreter

Reynaldo Martins

Operators from higher to low precedence

|  |  |
| --- | --- |
| Operator | Symbol |
| Negative signal | - |
| Division | / |
| Multiply | \* |
| Subtract | - |
| Sum | + |

Functions

|  |  |  |
| --- | --- | --- |
| Function | Arguments | Description |
| SQRT | X | Returns the square of X |
| CEIL | X | Returns the first integer above X |
| TRUNC | X | Returns the first integer bellow X |
| ROUND | X | Returns the rounding of X. (0.5 rounds to 1) |
| FACT | X | Returns the factorial of X |
| NEGAT | X | Returns the negative of X (the same as the operator Minus |
| ABS | X | Returns the absolute value of X |
| MOD | X,Y | Returns the rest of the division of X/Y. X and Y are truncated before been divided. |
| POW | X,Y | Returns X to the power of Y |
| MEAN | X1…Xn | Return the mean calculated over all parameters passed |
| MEDIAN | X1…Xn | Return the median of the set of all parameters passed |
| MIN | X1…Xn | Returns the lowest number of the set of all parameters passed |
| MAX | X1…Xn | Returns the greatest number of the set of all parameters passed |

Operation

1. Enter a valid expression using Number constants, Operators, (), Functions and Variables
2. Variables shall contain just alphabetical characters
3. The expression will be parsed and derived/”compiled”. In case the expression does not complies with the rules of the interpreter, expressed in the grammar used, a error message is informed
   1. The parsing stack as well as the derived/”compiled” stack is shown after both steps
4. After a successful parsing / compilation, the interpreter will request to inform the values of the variables found in the expression, if any
   1. The derived/”compiled” stack is shown where the variables were replaced by their values
5. The evaluation process starts. For each operation/function evaluation, the stack is updated and printed to be verified;
6. The final calculated value of the expression is the one which rests in the stack at the end of the process

Examples of expressions to be tested (to be tried – copy and paste)

1+3+2/4+5\*8+(7-5)\*10

(Median(1,3,8,200,10,80, 53,24,97 ) - Mean(1,3,8,200,10,80,53,24,97))\*(Max(1,3,8, 200,10,80,53,24,97)-Min(1,3,8,200,10,80,53,24,97)) + XVar + YVar\* XVar

Abs(-XVar-YVar)\*pow(2,8)/-4

Sqrt(XVar)\*ceil(1.7)+trunc(1.879)/round(1.65)+XVar\*(abs(5-7))

1----0004.60007000+5\*-0006

1--4.67+5\*-6

(2\*2+4)/(2\*(2+4))+100/XVar

Sqrt(256)/0

Sqrt(256)\*0\*10\*pow(2,8)

Grammar utilized in this simple mathematical interpreter

[Expression] -> [Expression-sumsubtract]

[Expression-sumsubtract] -> [Expression-sumsubtract ] + [Expression-multiply]

[Expression-sumsubtract] -> [Expression-sumsubtract ] - [Expression-multiply]

[Expression-sumsubtract] -> [Expression-multiply]

[Expression-multiply] -> [Expression-multiply] \* [Expression-divide]

[Expression-multiply] -> [Expression-divide]

[Expression-divide]-> [Expression-divide] / [Expression-simple]

[Expression-divide]-> [Expression-simple]

[Expression-simple] -> ( [Expression] )

[Expression-simple] -> - [Expression-simple]

[Expression-simple] -> Constant

[Expression-simple] -> Variable

[Expression-simple] -> Function ( [Expression-parameters-list] )

[Expression-parameters-list] -> [Expression-parameters-list] , [Expression]

[Expression-parameters-list] -> [Expression]

Remarks

* The recursive calls are done from less precedent to more precedent operators. It makes the evaluation stack to been built in a way that less precedent operators will not been evaluated unless those more precedents in each node been evaluated before.
* The grammar as above is not computational, since it will generate infinity recursive calls. A method is used as bellow to eliminate recursions at the left of a call

Strategy for eliminating recursions on the left side of the grammar

Original recursion [A] at the left :

[ A ] -> [ A ] X

[ A ] -> Y

Derivation solution :

[ A ] -> Y [ A-deriv]

[ A-deriv ] -> X [ A-deriv ]

[ A-deriv ] -> Empty

Grammar for a simple mathematical interpreter, after the derivations

[Expression] -> [Expression-sumsubtract]

[Expression-sumsubtract] -> [Expression-multiply] [Expression-sumsubtract -DERIV]

[Expression-sumsubtract -DERIV] -> + [Expression-multiply] [Expression-sumsubtract -DERIV]

[Expression-sumsubtract -DERIV] -> - [Expression-multiply] [Expression-sumsubtract -DERIV]

[Expression-sumsubtract -DERIV]-> Empty

[Expression-multiply] -> [Expression-divide] [Expression-multiply-DERIV]

[Expression-multiply-DERIV] ->\*[Expression-divide] [Expression-multiply-DERIV]

[Expression-multiply-DERIV] -> Empty

[Expression-divide]-> [Expression-simple] [Expression-divide-DERIV]

[Expression-divide-DERIV]-> / [Expression-simple] [Expression-divide-DERIV]

[Expression-divide-DERIV]->Empty

[Expression-simple] -> ( [Expression] )

[Expression-simple] -> - [Expression-simple]

[Expression-simple] -> Constant

[Expression-simple] -> Variable

[Expression-simple] -> Function ( [Expression-parameters-list] )

[Expression-parameters-list] -> [Expression] [Expression-parameters-list-DERIV]

[Expression-parameters-list-DERIV] -> , [Expression] [Expression-parameters-list-DERIV]

[Expression-parameters-list-DERIV] -> Empty