Term notations:

o++o versus today's arithmetic

Klaus Benecke

At present terms are treated too complicated. This paper wants to prove this statement and make a suggestion for improvement. The most important operations are addition (+) and multiplication (*). These operations have been around longer than the plus sign and the multiplication sign. Terms were formulated in sentences a long time ago.

```
2 and 3 times 4
```

Do they have the property that you evaluate them from left to right?

I strongly believe that this sentence was only several hundred years ago always answered with `yes`.

Today we would replace the above sentence with

```
2+3*4
```

instead.

If we look at this term independently of the problem above, we can first put 2 different pairs of parentheses here and get different results:

```
(2+3)*4 = 20
```

```
2+(3*4) = 14
```

These two results are mathematically secured. However, the assignment of one of two term values to

```
2+3*4
```

is not mathematics. It is a question of convention, i. e. primarily which concepts behind the two possibilities are more natural and which are easier to memorize or learn. Students of the lower grade and very many other people always tend to assume the first solution (They have not yet had the rules related to order of operations in school, or have forgotten them again, because these rules are very unnatural). Hence we should choose the first solution one, although at present the "unnatural convention" is still taught in school. The present chaos in this question should be eliminated this way, since also many simpler (more natural) calculators as well as the *Windows* calculator in the standard mode also determine 20.

Besides *o++o*, this also is applied in the programming language *Smalltalk*.

Today "still" usual notation for mathematical terms:

1. The binary operations +, *, -, :, mod are written infix; e. g. `1+2`.

2. The exponents of binary power and exponential functions are set higher; there is no symbol for this (e. g.: 3^4).
3. The root has a separate notation. The symbol √ with upper dash, which replaces the brackets.
4. The absolute value is expressed by two "|" signs, so that they replace also the brackets.
5. The unary factorial function (!) is written postfix; e. g.: `4! =24`
6. The remaining unary functions are written prefix; e. g.: `sin(3,1415)` and sometimes `sin 3,1415`
7. The remaining binary operations and operations of higher arity are written prefix with parentheses, with the input values separated by commas. 2 examples:
   `f(x+7,9)` (the binary function `+` infix and the binary function `f` prefix)
   `subtext("Today is Sunday",1,5)` (="Today");
8. Parentheses are evaluated first.
9. Factorial is evaluated next.
10. Often unary functions are evaluated next.
11. Power and root calculation is done before multiplication and division.
12. Multiplication and division is done before addition and subtraction.
13. Chained powers are calculated from right to left.
14. Other chained operations of the same order are calculated from left to right.
15. The term is written from left to right.

Magdeburg notation for mathematical terms:

1. All operations (functions) follow the first input value; if an operation has more than 2 input values, the further input values are separated by `!`.
2. Parentheses are evaluated first.
3. You are reading, writing, calculating and thinking from left to right.

Formal term definition in Magdeburg notation:

1. Every constant and every variable is a term.
2. If **t** is a term and **op1** is a unary operation symbol, then **t op1** is a term.
3. If **t1** and **t2** are terms and **op2** is a binary operation symbol, then **t1 op2 t2** is a term.
4. If **t1, t2, ..., tn** are terms and **op** is an n-ary operation symbol, then **t1 op t2!t3! ... !tn** is a term.
5. If **t** is a term, so is **(t)**

Criticism of today's usual arithmetic

1. The rules do not yet include structures like texts, lists, sets and tables.
2. Children and people, who had not used the precedence rules for a long time, always calculate `9` in the task `1+2*3`, just like *o++o*, *Smalltalk*, many pocket calculators and the *Windows* calculator in the mode normal.
3. A "long" calculation `21+52-92+3-7*2`... cannot be calculated by anyone in their head anymore. But as o++o term, because it is simply calculated from left to right. Therefore, one must always remember only one result until the next operation.

4. Today's notations given under 2 to 5 are not used in programming languages and are generally difficult to handle with the computer. The corresponding *o++o* notations are:

today's notation	Magdeburg Notation
3^4	`3 hoch 4`
`√3 +4`	`3 sqrt +4`
`|3+4| -5`	`3+4 abs -5`
`40!`	`1 .. 40 **`

5. The (German) comma notation of decimal numbers is not used in programming languages. For a binary function `f23(x,y)=2*x+3*y`, the expression `f23(3,4,5)` leads to problems, since `f23((3,4),5)` or `f23(3,(4,5))` may be meant. (*o++o*: `3.4 f23 5 or 3 f23 4.5`)
6. The rules for powers are used inconsistently in computer systems:
   `2^3^4 = (2^3)^4 = 4096` (*EXCEL* and *Libre Office Calc*)
   `2^3^4 = 2^(3^4) = 2.4178516e+24` (*Google Search*)
7. the rule for the unary minus is used inconsistently:
   `- 3 ^ 2 = (-3)^2 = 9` (*EXCEL* and *Libre Office Calc*)
   `- 3 ^ 2 = -(3^2) = - 9` (programming language basic calculator)
8. `10 - 3 + 2` is 9, not as the PEMDAS abbreviation wrongly suggests 5 (addition before subtraction)
9. In some computer systems `sin 3x = sin(3x)` and in others `sin(3)*x`
10. In *EXCEL* you cannot write `sin 3`, but have to use `sin(3)`.
11. Unary functions first means for example: 2+ sin pi+3*4=(2+(sin pi))+(3*4)
12. Because of multiplication/division before addition/subtration you have to put more brackets altogether, because
    `1+2*3 = 2*3+1`.

    In *o++o* you can replace
    `(1+2)*3` by `1+2*3` and
    `1+(2*3)` by `2*3+1`.
13. No semantics is provided for multi-line expressions. (However, a computer program usually goes over multiple lines).

    o++o:
    ```
    1+2+8
    *2+3
    = 55
    ```
    (here 4 brackets are saved over a RETURN)

For points 6 to 9 see also English Wikipedia "order of operations".

The following multi-line *o++o* program, that gives 2700, can be written in 2 lines, if you give the table the name `three_grandchildren.tab`:

```
<TAB!
NAME,  LOCATION, STIP l
Paul   Oehna     1000
Clara  Oehna      900
Sophia Dallgow    800
!TAB>
++
```

```
three_grandchildren.tab
++
```

Multi-line should always mean that you calculate from top to bottom. This would mean that the postfix notation for unary operations is already predetermined:

```
three_grandchildren.tab ++
```

Disadvantages of the Magdeburg notation

1. the polynomial notation like e. g. x^3+2x^2+3x+4 must be rewritten to

   `X+2*X+3*X+4`

   or

   `X poly [1 2 3 4]`

   or

   `X hoch 3 +(X hoch 2 *2)+(X*3)+4`

   The first notation has the advantage of being more efficient and the second the advantage of being easier to type in but the third requires more brackets.

   If the polynomial for X=87 is to be calculated, one must write only

   `87 poly 1 2 3 4`
2. Some people would have to change their way of thinking
3. Some of the mathematics books would have to be updated.

The following is a comparison of the two notations by examples, with other new symbols included whose meaning will be obvious from the context.

MD-notation	Conventional notation	Result or remark
`1+2*3`	`(1+2)*3`	9
`2*3+1`	`2*3+1`	7
`2 sin`	`sin 2 or sin(2)`	0.909297426826
`-4 abs sqrt`	`sqrt(| -4|)`	2.
`3+4*5+6`	`(3+4)*5+6`	41
`1 4 7 4 ++` (= `[1 4 7 4] ++`)	`1+4+7+4` or `sum([1;4;7;4])`	16

2.30 3.72 4.77 *1.19	2.30*1.19,3.72*1.19, 4.77*1.19	2.737 4.4268 5.6763
3 .. 6 **	6!/2!	360
-16 abs sqrt sqrt	sqrt(sqrt(|-16|))	2.
37 hoch 3	37 ^ 3 or 37^3	50653
37 poly [1 2 3 4]	37^3+2*37^2+3*37+4	53506
1 2 2 1 3 1 ++: rnd 1	rnd(avg([1 2 2 1 3 1]),1)	1.7
pi+3*2 cos abs sqrt	sqrt(abs(cos((pi+3)*2)))	0.979882792302
X sin + (2 cos) sqrt	sqrt(sin(x) + cos(2))	
-1 ... 5!0.000001 sin max		approximate maximum of sine function in interval [-1, 5]
0 ...pi!0.001 cos abs *0.001 ++		approximate determined integral of |cos(x)|
1+2 *3+4	(1+2)*(3+4)	21
1*2 +3*4	1*2+3*4	14

Because of the described problems I suggest to introduce the Magdeburg notation. It is to be noted that *o++o* calculated 2 years ago also in the sense of the today's school mathematics, so that o++o could be reset also again. However, this would significantly worsen the readability of our programs.

```
0 ...pi!0.0001 cos abs sqrt *0.0001 ++
```

would then have to be written in the following way:

```
++(sqrt(abs(cos(...(0,pi,0.0001))))*0.0001)
```

or even

```
++(sqrt(|cos(...(0,pi,0.0001)|)*0.0001)
```

Many newspapers and other media also deal with mathematical puzzle problems. In Facebook it is stated that many adults are not able to solve the task

```
8 ÷ 2 x (2 + 2)
```

correctly. The same confirms a German newspaper for the task

```
9 - 3 : 1/3 + 1
```

In the first task it is stated that dot before dash calculation is valid, but for ÷ and x there is no corresponding rule. Therefore, one must calculate from left to right. Because of this, we conclude that the last statement is currently taught too little in school lessons. One must often repeat "dot before dash arithmetic", so that less time remains for the more important rule "from left to right".

Furthermore, the understanding of "dot before dash arithmetic" is made more difficult when the dot is an x and the division sign contains a dash.

Many people do not know which is the multiplication sign on the computer keyboard. They often choose the letter x.

We cannot teach resp. learn as much as we want. The human brain has limited capabilities.

Therefore, I believe that one should also throw many known mathematical notations overboard in school, if one seriously pursues the goal that everyone should learn a programming language.

- comma notation for decimal numbers
- absolute strokes
- factorial sign
- root sign
- power notation
- polynomial notation
- ...

Acknowledgment:
I would like to thank Eicke Redweik for valuable remarks.

Summary:
The concepts that require the least amount of learning, that are not easily forgotten, and whose representations are most readable and understood by the computer and for which standards exist will prevail.