Comments on School Education in Mathematics and Computer Science

KLAUS BENECKE
beneckeSysteme
Schröders Garten 5,
39175 Gerwisch,
GERMANY

Abstract: - This work aims to improve problem-solving skills even for students and adults, who are not so interested in mathematics. This requires simple algorithms. The efficiency of these is not the focus of this work, as we assume that computers will continue to take over this computational work. We concentrate on important, flexible operations that are as universal as possible. In doing so, we rely on the object of the structured table. A structured table can represent a number of any type, but also texts and structured collections of text and numbers. This means that the focus is no longer only on individual data operations such as addition, multiplication, sine, root, etc., but also on mass data operations such as selecting (choosing), sorting, aggregating, restructuring, joining (merging tables), etc. The work clearly shows that these operations can also be used to develop new, simpler solutions for the problems now being dealt with at school. Simpler, more universal algorithms that can be programmed by anyone should be taught at school. The work on the data model o++o was strongly motivated by the Relational data model with its language SQL and the language CONVERT, which tried to extend the flat table model to hierarchical data structures.

Key-Words: - data model, structured tables, math lessons, simple, easy-to-understand and easy-to-program algorithms, tabment, o++o, end-user language, algebraic mass-data operations, school.

Received: April 22, 2024. Revised: November 16, 2024. Accepted: December 18, 2024. Published: January 14, 2025.

1 Introduction

This paper attempts to show that students' problem-solving skills could be improved by using the data model o++o for tabments (structured TABles + structured docuMENTs) in school. Since computers exist today, it is no longer necessary to teach the most efficient algorithms. Simple algorithms are often easier to understand, but often more inefficient. However, this is no longer a disadvantage in the age of digitalization. Simple algorithms are of course also easier to program. Anyone who can convert a problem into computer code or understand a code can also better understand and interpret the computer results and verify the correctness of the result. If necessary, they can improve an incorrect or undesirable result by modifying the corresponding computer program. If this is possible without additional programmers, programming efficiency increases considerably. We put forward the following hypotheses:

1. Understanding basic arithmetic operations is very important for people's ability to think logically.
2. Today, we believe that less than one percent of people are able to program the decimal number algorithms of basic arithmetic operations.
3. Anyone who can program an algorithm in a computer language has a better understanding of it.
4. The decimal number algorithms for multiplication and division are too complicated. If you don't perform them for several decades, you often forget them.
5. The (p,q) formulas taught at school for determining the zeros of second-degree polynomials are too specific and should be replaced by more universal, simpler algorithms for determining the zeros of more general functions.
6. Teachers and students can manage the marks themselves.
7. Selection is a very important operation that is not yet given enough attention in school mathematics. It is also used sensibly in number theory below.

Conclusion: Simpler, more universal algorithms that can be programmed by anyone must be taught at school. The theoretical foundation for o++o is the algebraic specification language published in [1], [2]. With o++o an attempt was made to generalize

the Relational data model [3], [4] and SQL [5], [6]. o++o uses structured tables, which were called in CONVERT [7] hierarchical. By this data model, the development of o++o was motivated, too. One of the most important operations of the data model o++o, the stroke-list-operation stands behind the *gib*-statement. A formal specification of it is published for example in [8]. The *restruct*-operation of [9] had similar goals, but *restruct* is not so expressive as the *gib*-statement. By *gib* additionally for example aggregations can be realized. The approach on o++o was generalized to documents with the dawn of XML and XQuery, [10]. Problems mentioned in [11] like "false drop" were important for our development. The ideas of formalization of [12] influenced also our early motivation.

2 Counting, a Basic Skill

Counting is the simplest operation considered here, so we do not need to go into it in detail. Counting means adding one for each new sub-object, hence the notation ++1. ++ (sum) stands for many plus signs.

count.otto: Count some names

```
Thor Jens Peter Nikolaus Bernd Gerd Siegfried
++1
```

Result (tab)

```
ZAHL
7
```

3 Addition

The addition can be carried out in o++o in the same way as with any calculator. However, in order to better understand the operation, further algorithms and operations will be presented.

add_term.otto: seven plus eight

```
7+8
```

Result (tab)

```
ZAHL
15
```

Column names are assigned using assignments (:=). They can make comments superfluous. You cannot do without them for more complex problems.

add_tags.otto: seven plus eight

```
BOYS:=7
GIRLS:=8
PEOPLE:=BOYS+GIRLS
```

Result (tab)

```
BOYS,GIRLS,PEOPLE
7    8     15
```

In the following, *pred* denotes the predecessor, *l* stands for list, and *x .. y* generates all numbers from x to y. By the first line, each of the numbers 1 to 8 is tagged by SU1. *next* can be considered as a binary operation. Binary operations are always written infix.

add_next.otto: seven plus eight

```
SU1l:= 1 ..8
SU  := 7+1 next SU pred +1 at SU1
```

Result (tab)

```
SU1,SU 1
1    8
2    9
3   10
4   11
5   12
6   13
7   14
8   15
```

The while loop in o++o has been somewhat syntactically sugar-coated. The first column does not need to be programmed. It always contains the values 1 2 3 ... . *while* is a ternary operation. The second and third argument are separated by *!*. If a program-line starts with more than 3 blanks, then it is a logical part of the preceding line.

add_while.otto: seven plus eight

```
SU1,SU 1:= 7+1 while SU1 <=8 !
           SU pred +1
```

Result (tab)

```
SU1, SU 1
1    8
2    9
3   10
4   11
5   12
6   13
7   14
8   15
```

The *+coll* operation is a "union operation". It is simpler than addition. It is applied here to the two lists below. These two lists are simply merged. No duplicates are removed, as with the set-theoretical union.

add_union.otto: seven plus eight

```
Thor Jens Peter Nikolaus Bernd Gerd Siegfried
+coll Freya Lynn Gisela Marie Clara Sieglinde
      Sophia Brunhilde
++1
```

Result (tab)

```
ZAHL
15
```

4 Multiplication

mult_term.otto: seven times eight
7*8
Result (tab)
ZAHL
56

The list of boys is assigned the name *BOYl* (l abbreviated list) below. This assigns the name (tag) *BOY* to each boy. The most important part of the following program is the keyword *at*. This appends the list of girls to each boy. This means that each girl appears seven times. By *gib GIRLl*, these seven lists are combined into one list with duplicates and can then be counted without any problems.

mult_simple.otto: seven times eight (names)
BOYl:= Thor Jens Peter Nikolaus Bernd Gerd Siegfried GIRLl:= Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde at BOY gib GIRLl ++1
Result (tab)
ZAHL
56
The intermediate result after the first two program lines (tabh).
BOY, GIRLl l
Thor Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde Jens Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde Peter Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde Nikolaus Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde Bernd Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde Gerd Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde Siegfried Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde

The third multiplication program presents a simplified *gib*-instruction. Below, only the count aggregation is executed. The corresponding elements are counted for one column of each hierarchy level.

mult_simple_gibagg.otto: seven times eight
BOYl := Thor Jens Peter Nikolaus Bernd Gerd Siegfried GIRLl:= Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde at BOY gibagg ++1
Result (tab)
BOYCNT,GIRLCNT
7 56

The operation **coll* is rarely used in o++o. However, this "cross product" (Cartesian product) is presented here for the sake of completeness.

mult_cart_prod.otto: seven times eight
Thor Jens Peter Nikolaus Bernd Gerd Siegfried *coll Freya Lynn Gisela Marie Clara Sieglinde Sophia Brunhilde ++1
Result (tab)
ZAHL
56
Intermediate result after the second line (tab) (excerpt)
WORT, WORT l
Thor Freya Thor Lynn Thor Gisela Thor Marie Thor Clara Thor Sieglinde Thor Sophia Thor Brunhilde Jens Freya Jens Lynn . . . Gerd Brunhilde Siegfried Freya Siegfried Lynn Siegfried Gisela Siegfried Marie Siegfried Clara Siegfried Sieglinde Siegfried Sophia Siegfried Brunhilde

*x *l y* turns x into a list of y x-objects. In many programming languages, the operation is called *make*.

mult_mal_l_cnt.otto: seven times eight

```
BOYl:=Thor Jens Peter Nikolaus
      Bernd Gerd Siegfried
*l 8
gib BOYl
++1
```

Result (tab)

```
ZAHL
56
```

The intermediate result after the second line (tabh)

```
BOYl l
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
Thor  Jens  Peter  Nikolaus  Bernd  Gerd
Siegfried
```

mult_mal_l.otto: seven times eight

```
7 *l 8
++
```

Result (tab)

```
ZAHL
56
```

The intermediate result after the first program line (tabh)

```
ZAHLl
7 7 7 7 7 7 7 7
```

In the following, the first level of the *BOYl l* table does not contain a column name, which is why only one column appears in the overall result.

mult_mal_l_gibagg.otto: seven times eight

```
BOYl:= Thor Jens Peter Nikolaus Bernd
       Gerd Siegfried
*l 8
gibagg ++1
```

Result (tab)

```
BOYCNT
56
```

In ancient Egypt and Babylon, multiplication was reduced to addition, using an efficient doubling algorithm.

mult_babylon.otto: 18 * 117

```
NR,F1,POT2 l:=18,1 while POT2 <= 117 !
              preds+preds
gib F1,POT2 l-
POT2SUM,F1SUM:=(POT2,F1) next
          (preds + (POT2,F1))
          if POT2SUM pred +POT2<=117!
          preds at POT2
```

Result (tab)

```
F1 , POT2 ,POT2SUM,F1SUM l-
1152 64     64     1152
 576 32     96     1728
 288 16    112     2016
 144  8    112     2016
  72  4    116     2088
  36  2    116     2088
  18  1    117     2106
```

The number in the last column and last row represents the final result. *preds* abbreviates the tuple of the predecessor. In the above while-statement, it stands for *(F1 pred, POT2 pred)* and in the last line it abbreviates (*POT2SUM pred, F1SUM pred*)

Now follows the multiplication program that comes closest to written multiplication. However, the operations used (*mat and cross) have probably a broader application in mathematics and IT practice.

mult_mat_cross.otto: 117 *18

```
100 10 7 *mat (10,8)
cross ++
```

Result (tab)

```
ZAHL ,ZAHL ,SUM? l
1000  800   1800
 100   80    180
  70   56    126
1170  936   2106
```

It should be noted that the list *100 10 7* can be generated from 117:

```
117 cut 1 zahl * (100 10 1)
```

1 10 100 is obtained as follows:

```
Xl:= 0 ..2
XPOT:=10 ^ X
```

and (10,8) results from

```
10 8 transpose
```

Finally, we remark that the operation *cross* can be omitted:

```
100 10 7 *mat (10,8) ++
```

gives also the product.

The following program is also important, as a broadly usable concept is also applied here. The *next*-operation below adds a new column GIRLS to

the BOY column. The first value is determined by the first expression (8) and all others result from the predecessor (pred) by adding 8 in each case. The last row is not selected here.

mult_next.otto: seven times eight
BOYl:= Thor Jens Peter Nikolaus Bernd Gerd Siegfried GIRLS := 8 next GIRLS pred + 8 at BOY
Result (tab)

BOY,	GIRLS l
Thor	8
Jens	16
Peter	24
Nikolaus	32
Bernd	40
Gerd	48
Siegfried	56

The following *while*-loop allows a compact formulation of multiplication. Two columns are introduced here at the same time. The first column BOY takes the values 1, 2, 3, ... etc. This has been set by default. This "syntactic sugar" means that the calculations for the first column no longer need to be specified. The first value of the GIRLS column is again derived from 8 and all other values are determined using the expression *GIRLS pred +8*. The calculation takes as long as the while condition *BOYS <= 7* is fulfilled.

The notion “syntactic sugar“ describes not an essential new concept. It is introduced to simplify the application of the concept.

mult_while.otto: seven times eight
BOYS,GIRLS l := 8 while BOYS <= 7 ! preds +8
Result (tab)

BOYS	GIRLS l
1	8
2	16
3	24
4	32
5	40
6	48
7	56

5 Division

The following program for a division is essentially the same as the program above. Only the while conditions are different and the result is in the first column.

int_div.otto: fifty-six divided by eight
BOYS,GIRLS l := 8 while GIRLS <= 56 ! preds +8
Result (tab)

BOYS	GIRLS l
1	8
2	16
3	24
4	32
5	40
6	48
7	56

6 Extrema

The following are simple examples of maximum aggregation.

maximum1.otto
1 3 7 3 5 max
Result (tab)

ZAHL
7

maximum2.otto: Calculate the maximum of three numbers of different types.
4/7 0.53 0.54 max
Result (tab)

RATIO
4/7

In the following, maxima are formed not only from lists of numbers, but also from tuples of expressions. Since the comma is a "normal" operation that is calculated from "left to right", you have to be careful. The last addition of 0.11 is applied to each component of the triple 1.2,1.,1.1. This results in the triple 1.31,1.11,1.21. The first component is then the maximum.

maximum3.otto: Maximum of comma-separated expressions
1+0.2, 1st, 1.1+0.11 max
Result (tab)

PZAHL
1.31

The semicolon is also used to form pairs. However, it separates more sharply, which is why the last addition is only applied to the third component.

maximum4.otto: Maximum of expressions when a semicolon is used
1+0.2, 1.; 1.1+0.11 max
Result (tab)
PZAHL
1.21

By *1* .. *10* the numbers 1 2 3 4 5 6 7 8 9 10 are generated. The *sin*-function is applied to each of the numbers. The maximum of these 10 function values is then output.

maximum5.otto: Maximum of ten function values
MAX10SINE:=1 .. 10 sin max
Result (tab)
MAX10SINE
0.989358246623

In contrast to ".." , "..." is a ternary function. Below, "..." has the input values "1" (start value), "2" (end value) and "0.000'1" (increment). The *sin*-function is applied to each of the 10'000 numbers generated and the largest is then output. It represents an approximation for the local maximum of the *sin*-function in the interval [1;2]. The apostrophe is used to make numbers easier to read. It is an idea of Switzerland.

maximum6.otto: Approximation of a local maximum
LOCAL_MAX_SINE:=1 ... 2! 0.000'1 sin max
Result (tab)
LOCAL_MAX_SINE
0.999999999993

Now the same program is applied to the parabola "$-X^2 + X$" in the interval [-1;3]. The coefficients -1 1 0 could also have been written in brackets [-1 1 0].

maximum7.otto: Local maximum of the parabola "-X^2 + X" opened downwards
LOCAL_MAX_PARABOLA := -1 ... 3!0.0001 poly -1 1 0 max
Result (tab)
LOCAL_MAX_PARABOLA
0.25

maximum8.otto: Maximum of the third-degree polynomial "-X^3 + X^2" in the interval [0;3]
LOCAL_MAX_POLYNOMIAL_GRADE3:=0 ... 3!0.1 poly [-1 1 0 0] max
Result (tab)
LOCAL_MAX_POLYNOMIAL_GRADE3
0.147

We recognize that we can determine local extreme values without understanding differential calculus.

7 Area and Perimeter Calculations

circle.otto: Area and circumference of a circle with radius 2.34
AREA:= 2.34 ^ 2 * pi PERIMETER:= 2.34 * 2 * pi
Result (tab)
AREA , PERIMETER
17.202104734 14.7026536188

circles.otto: Area and circumference of 3 circles
RADIUSl := 2.34 3.456 44.999 AREA:=RADIUS ^ 2 * pi PERIMETER:=RADIUS * 2 *pi rnd 3
Result (tab)
RADIUS, AREA, PERIMETER l
2.340 17.202 14.703 3.456 37.523 21.715 44.999 6361.442 282.737

rectangle.otto: Area and perimeter of a rectangle
AREA:=2.34 *5.67 PERIMETER:=2.34 + 5.67 * 2
Result (tab)
AREA, PERIMETER
13.2678 16.02

Note in the above PERIMETER formula that no brackets are required, as the calculation is simply from left to right.

rectangles1.otto: Area and perimeter of four rectangles with given edge lengths

```
<TAB!
A,   B  l
2.34 5.50
3.45 6.60
6.78 7.70
9.99 8.80
!TAB>
AREA := A * B
PERIMETER := A + B * 2
rnd 2
```

Result (tab)

```
A ,  B ,  AREA ,PERIMETER l
2.34 5.50 12.87 15.68
3.45 6.60 22.77 20.10
6.78 7.70 52.21 28.96
9.99 8.80 87.91 37.58
```

rectangles2.otto: Rectangle calculations, where the edges are given in separate lists. Both four-element lists are connected by a unary position join.

```
Al:= 2.34 3.45 6.78 9.99
Bl:= 5.5  6.6  7.7  8.8
posjoin2
AREA := A * B
PERIMETER := A + B * 2
rnd 2
```

Result (tab)

```
A,   B,   AREA, PERIMETER l
2.34 5.50 12.87 15.68
3.45 6.60 22.77 20.10
6.78 7.70 52.21 28.96
9.99 8.80 87.91 37.58
```

After *posjoin2*, the sub-table is created with the header *A,B l* of the final result. posjoin2 is an abbreviation for posjoinposjoin (analogous to + and ++).

In the following program, the three side lengths of a triangle are given.

triangle.otto: Calculation of the area of a triangle with 3 given side lengths

```
AREA := 2.34 3.45 4.56 areatriangle rnd 2
```

Result (tab)

```
AREA
3.95
```

triangles.otto: Calculation of two triangular areas given the corner coordinates

```
<TAB!
AX, AY, BX, BY, CX, CY l
1   2   3   5   5   6
2   3   4   5   2   7
!TAB>
C := AX - BX ^ 2 + (AY - BY ^ 2) sqrt
B := AX - CX ^ 2 + (AY - CY ^ 2) sqrt
A := BX - CX ^ 2 + (AY - CY ^ 2) sqrt
AREA:= A,B,C transpose areatriangle
rnd 2
```

Result (tab)

```
AX ,AY ,BX ,BY ,CX ,CY ,C,    B ,   A ,
AREA l
1   2   3   5   5   6   3.61 5.66 4.47
8.06
2   3   4   5   2   7   2.83 4.00 4.47
5.57
```

Area_and_perimeter_circle.otto: Approximation of half the area and half the circumference of a circle with a radius of 1

```
Xl:= 0 ...1!0.000'001
Y:= X poly [-1 0 1] sqrt
RECTANGLE:= Y *0.000'001
SECANT := Y - Y succ ^ 2 +
          (0.000'001 ^ 2) sqrt
gib AREA,PERIMETER
    AREA:=RECTANGLE!++
    PERIMETER:=SECANT!++
*2
```

Result (tab)

```
AREA,          PERIMETER
1.57079732621 3.1415926533
```

The above calculation requires relatively large amounts of resources as it has not been optimized, contrary to examples maximum5 – maximum8. Here, we rely on fast computers. (Sub-) Terms are optimized, if they start with ".." or "…" and end with an aggregation like *max*, *min*, ++, ** etc.. In this case classical loops are used, internally.

area_and_length_of_rope.otto: Length of a hanging rope and area under the rope.

```
Xl:= -2 ...2!0.000'1
Y:= e ^ X + (e ^ (X* -1)) * 0.5
RECTANGLE:= Y *0.000'1
SECANT := Y - Y succ ^ 2 +
          (0.000'1 ^ 2) sqrt
gib AREA,LENGTH
    AREA:=RECTANGLE!++
    LENGTH:=SECANT!++
```

Result (tab)

```
AREA,          LENGTH
7.25409704131 7.25372081461
```

Result of the following "subprogram" (strucdiagram bar)

```
Xl:= -2 ...2!0.01
Y:= Y:= e ^ X + (e ^ (X* -1)) * 0.5
X::=X text
```

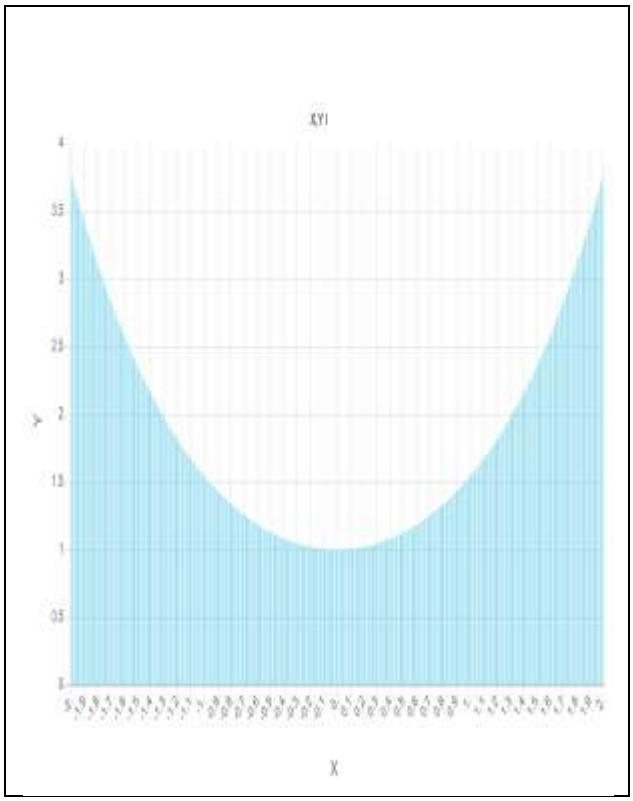

Fig. 1: Bar diagramm of a hangig rope

The X values had to be converted into text so that they are not interpreted as additional columns but as signatures. This diagram shows a large number of rectangular columns. They illustrate that the blue area is approximately filled by the rectangles. The *cosh*-function can be interpreted as a sagging rope. 7.25 is then the length of the rope.

8 Zero Determination

integer_zero.otto: Find the integer zeros of the parabola "X^2-9X+20" in the interval [-100;100]
Xl:= -100 .. 100 Y:=X poly [1 -9 20] sel Y=0
Result (tab)
X ,Y l
4 0 5 0

This program is probably methodologically valuable as it directly implements the definition of the zero. The fact that not every zero can be determined in this way due to the limitations of the number ranges in the computer must be explained to the pupils. We consider the determination of zeros for parabolas according to the formulas:

X1 = p:2 ^ 2 - q sqrt - (p:2)

X2 = p:2 ^2 - q sqrt * -1 - (p:2)

are not universal enough and too complicated, so that they could possibly be removed from some of the school curriculums. In the following, we will not give the programs for the zeros according to the regula falsi, the Newton method and the bisection method, although they can be used relatively universally and are also efficient. On the other hand, we want to give even simpler algorithms. They have never been mentioned in the past due to their inefficiency.

zeropoint.otto: Determine the zero of the sine function in the interval [3;4]
Xl:= 3 ...4 !0.000'001 Y:=X sin sel Y >0 last
Result (tab)
X , Y l
3.141592 6.53589793076e-007

zero.otto: Determine the zeros of the sine function in the interval [3,10]
Xl:= 3 ...10 !0.000'01 Y := X sin sel Y succ * Y <=0
Result (tab)
X , Y l
3.14159 2.65358979335e-006 6.28318 -5.30717958669e-006 9.42477 7.9607693791e-006

If, for example, the first zero point is to be determined even more precisely, you can enter a very small interval around this approximation and reduce the step size accordingly:

zero_more_excact.otto: Improve the accuracy of the first zero digit
Xl:= 3.14159 ...3.1416 ! 0.000'000'001 Y := X sin sel Y succ * Y <=0
Result (tab)
X , Y l
3.141592653 5.89793225706e-010

It is obvious that the above two programs can also be applied to polynomials of third and higher degree and are therefore more universally applicable than the (p,q)-formula.

pythagoras_hypo.otto: Compute the hypothenuse with the formula $a^2 + b^2 = c^2$

```
A:=  3
B:=  5
Cl:= 3 ...7!0.000'01
PYTHAGORAS:= C,A,B ^2 --
sel PYTHAGORAS succ *PYTHAGORAS <=0
```

Result (tab)

```
A ,B ,(C       ,PYTHAGORAS  l)
3  5   5.83095 -2.20975000076e-005
```

The hypothenuse has approximately a length of 5.83095. In the computation, we do not need a square root. -- subtracts all components from the first one.

pythagoras_kathe.otto: Compute the Kathe with the formula $a^2 + b^2 = c^2$

```
A := 3
Bl:= 3 ...7!0.000'01
C := 6
PYTHAGORAS:= C,A,B ^2 --
sel PYTHAGORAS succ *PYTHAGORAS <=0
```

Result (tab)

```
A ,(B       ,PYTHAGORAS l),     C
3   5.19615 2.51774999924e-005 6
```

percentage_calculation.otto: o++o offers four percent operations: % +% -% and net

```
GROSS:=200
NETl:= 150 ...200!0.000'1
F:=NET +% 19 - 200
sel F succ * F <0
OTTONET:=GROSS net 19
GROSS2:=OTTONET +% 19 at GROSS
MINUS19PROZ:=GROSS -% 19
```

Result (tab)

```
GROSS ,GROSS2 ,OTTONET       ,MINUS19PROZ
         ,(NET     ,F      l)
200    200.    168.067226891 162.
           168.0672 -3.20000000045e-005
```

We believe that it is not sufficient to provide only the simple functions +% and -%. Often the question arises to compute the net value, if the gross value and the tax are given. o++o offers here the operation *net*, such that the user does not need to compute zeros or other more complicated formulas.

9 Number Theory

In this very short section, we will also restrict ourselves to particularly simple algorithms.

lcm.otto: The least common multiple of 12 and 8

```
Xl:= 1 .. 12 *8
sel X rest 12 = 0
min
```

Result (tab)

```
ZAHL
24
```

gcd.otto: The greatest common divisor of 14 and 21

```
Xl:= 1 .. 14
sel 21 rest X = 0 & 14 rest X = 0
max
```

Result (tab)

```
ZAHL
7
```

x rest y is an abbreviation for *x divrest y nth 2*.

sieve_eratosthenes.otto: Find all prime numbers under one hundred.

```
Xl:=2 .. 50
Yl:=2 .. 10 at X
PROD:=X*Y
gib PRODm #set of all products
PRIMl:= 2 ..100
sel- PRIM in PRODm
proj- PROD
```

Result (tabh)

```
PRIMl
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
53 59 61 67 71 73 79 83 89 97
```

10 o++o-Proofs

Mathematical proofs play a minor role in today's school and outside the world of mathematicians. Nevertheless, everyone needs confidence in the results of the calculator or pocket calculator. When someone is confronted with a new calculator or system, the first thing they do, is set problems, whose solution they already know, e.g. *2 times 3*.

commutativity.otto: "Prove" the commutative law of multiplication for natural numbers.

```
Xl:= 1 .. 1'000
Yl:= 1 .. 1'000 at X
sel- X*Y=Y*X
```

Result (tab)

```
X ,Yl l
```

Obviously not all the numbers have been calculated here. For a third class, however, this could be satisfactory.

avg_proof.otto: "Proof" that ++: can simply be traced back to ++ and ++1.

```
RANDOMNR1:=1 ..x 10!10
X1:=1 ..x RANDOMNR!RANDOMNR
SUM:=X1 ++
CNT:=X1 ++1
OTTOAVG:=X1 ++:
MYAVG:=SUM:CNT
#sel- MYAVG=OTTOAVG
```

Result (tab)

```
RANDOMNR ,SUM ,CNT ,OTTOAVG,
    MYAVG,         X1 1
 4         10    4    2.5
    2.5             3 2 2 3
10         56   10    5.6
    5.6             2 5 5 4 6 8 10 8 2 6
 9         52    9    5.77777777778
    5.7777777777  8 9 2 3 7 2 6 5 9 9
 2          2    2    1.
    1.             1 1
 6         20    6    3.33333333333
    3.33333333333  1 5 4 4 4 2
 1          1    1    1.
    1.             1
 2          4    2    2.
    2.             2 2
 3          4    3    1.33333333333
    1.33333333333 2 1 1
 6         21    6    3.5
    3.5             3 3 6 1 5 3
 7         27    7    3.85714285714
    3.85714285714   1 1 3 7 4 5 6
```

If the selection (last line) is activated, the first line could be improved:
`RANDOMNR1:=1 ..x 1000!1000`
and receives the empty list of the above type.

11 Marks Management

As is well known, the average of numbers is calculated as the quotient of the sum of the numbers and the number of numbers. ++: is therefore an abbreviation for ++:++1.

avg.otto: average of several marks.

```
AVERAGE := 1 2 3 2 1 ++:
```

Result (tab)

```
AVERAGE
1.8
```

Table 1. marks1.tabh is a tabment in "horizontal tab" format

marks1.tabh

```
SUBJECT, MARK1 1
Math      1 2 3 2
Phy       1 3 5 3 1
Eng       5 4 3 1 2
```

The following four programs use Table 1.

avg_all.otto: Average of all numbers in a table

```
marks1.tabh ++:
```

Result (tab)

```
PZAHL
2.57142857143
```

avg_avg.otto: Averages for all subjects and the average of these averages.

```
marks1.tabh
AVGSUBJECT:=MARK1 ++:
AVGTOTAL:= AVGSUBJECT1 ++:
rnd 1
```

Result (tabh)

```
AVGTOTAL ,(SUBJECT ,AVGSUBJECT ,MARK1 1)
2.5        Math     2.0          1 2 3 2
           Phy       2.6          1 3 5 3
1
           Eng       3.0          5 4 3 1
2
```

total.otto: Set the average and the mean absolute deviation from the average "below" each collection.

```
marks1.tabh
total ++:,mad
rnd 1
```

Result (tabh)

```
SUBJECT ,MARK1 1
Math      1 2 3 2 2.0 0.5
Phy       1 3 5 3 1 2.6 1.3
Eng       5 4 3 1 2 3.0 1.2
avg       2.6
mad       1.1
```

totalhierar.otto: Calculate two aggregation types of the marks in both parent hierarchies. MARK is the only numeric column in the given table, so only MARK aggregations are calculated.

```
marks1.tabh
totalhierar ++:,mad
rnd 1
```

Result (tabh)

```
MARKAVG   ,MARKMAD   ,(SUBJECT   ,MARKAVG2
,MARKMAD2 ,MARK1 1)
2.6          1.1           Math         2.0
```

```
0.5        1 2 3 2
                    Phy                    2.6
1.3        1 3 5 3 1
                    Eng                    3.0
1.2        5 4 3 1 2
```

cross.otto: Calculate the grade averages horizontally and vertically.

```
<TABH!
SUBJECT,ERNST1,CLARA1,SOPHIA1,ULRIKE1,KAETHE1,CLAUDIA1 1
Math    2 3 2  1 1 2  1 3 2   1 2 3 3 1 2     1 1
Physics 2 3    4 2    1 2     2 5     1 1     2 3
German  1 1           3 5     4
!TABH>
cross ++:
rnd 1
```

Result (tabh)

```
SUBJECT,ERNST1 ,CLARA1 ,SOPHIA1 ,ULRIKE1 ,KAETHE1 ,CLAUDIA1 ,AVG? 1
Math      2.3     1.3     2.0      2.3      1.5      1.0      1.8
Physics   2.5     3.0     1.5      3.5      1.0      2.5      2.3
German    1.0             4.0      4.0                        2.8
avg       2.0     2.0     2.4      2.9      1.3      1.8      2.1
```

Result if `cross` is replaced by `cross+` (tabh)

```
SUBJECT ,ERNST1 ,  CLARA1 ,  SOPHIA1 , ULRIKE1 ,   KAETHE1 ,CLAUDIA1 ,AVG? 1
Math     2 3 2 2.3 1 1 2 1.3 1 3 2 2.0 1 2 3 3 2.3 1 2 1.5  1 1 1.0   1.8
Physics  2 3 2.5   4 2 3.0   1 2 1.5   2 5 3.5     1 1 1.0  2 3 2.5   2.3
German   1 1 1.0             3 5 4.0   4 4.0                         2.8
avg      2.0       2.0       2.4       2.9        1.3      1.8       2.1
```

The given table seems to represent the grades of a gradebook in a compact way. If all subjects and names already exist in this table, no new subject and no new name need to be inserted when inserting a grade, even if the corresponding grade list is still empty. A hierarchical table of the type *SUBJECT,(NAME,MARKl m)m* contains each subject only once, but the names must be inserted several times. If this amount of data is stored in a flat table *SUBJECT,NAME,MARK l*, even more redundancy is created and even more memory space is wasted. In a flat set (relation of the relational data model), a time or position column would also have to be introduced, which makes the whole thing even more unwieldy and inefficient.

The columns and rows in the table above could also be swapped.

12 Conclusions and Future Work

We tested o++o already 10 years ago successfully in a seventh class of a Gymnasium. But, in the last years, we improved and added many operations and concepts, such that we hopefully will have yet greater success in teaching and applying o++o.

Then Chi-square test has to be included, for example. Although o++o algorithms seem to be simpler than the algorithms taught in school now, it will probably take many years until o++o-algorithms will be a part of the curricula of schools. So, o++o has to be taught in schools, probably it can influence and improve the lessons of all classes from 1 to 12.

Nevertheless, there remains also a lot of implementational work. First of all, o++o has to be put on top of Relational and NoSQL database systems. Here, some optimization techniques are needed. This requires for example a good understanding of selection- and join- operations. In the Relational data model, it is possible to commute content-oriented conditions always. This does not hold for selections in structured tables, such that more sophisticated rules have to be applied.

o++o-programs for extrema of or areas under functions seem to be very inefficient, because of large amounts of storage area, which is needed for a statement like `0 … 10!0.000'001`. This storage area is not needed, if the statement ends with an aggregation like ++ or *max*, for example. We optimized such subexpressions already by loops.

In which age groups of pupils o++o algorithms are useful?

Examples of stroke-list-operation (gib) can be taught on paper already for preschool children. Table loops probably in lower school and the application of integral and differential calculus in 9 class, because you don't have to understand these complicated theories.

Acknowledgment:
I thank Heinz Kaphengst and Horst Reichel for the development of the algebraic specification language for partial operations and Rüdiger Achilles for valuable remarks to this paper. Further, thanks are directed to the following computer experts for their valuable contributions to the older implementations of the o++o system: Wolfgang Reichstein, Dmitri Schamschurko, Martin Schnabel, Andreas Hauptmann. Stephan Schenkl and Eicke Redweik worked for the project from 2019-2021 "Intelligent Analysis and Visualization of German Wikipedia", which was funded by EFRE EU and IB Sachsen-Anhalt.

References:

[1] H. Kaphengst, H. Reichel, "Algebraische Algorithmentheorie", *VEB Robotron*, Wiss. Informationen und Berichte, Nr. 1/71 Reihe A, Sommer 1971.

[2] H. Reichel *"Initial Computability, Algebraic Specifications, and Partial Algebras"*, Claredon Press, Oxford, UK, 1987.

[3] E.F.Codd, „A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, Vol. 13, No. 6 June 1970, S. 377-387.

[4] E.F. Codd, „Extending the Database Relational Model to capture more Meaning“, *ACM Transactions on Database Systems*, Vol. 4 , No 4, Dec. 1979, S. 379-434.

[5] Donald D. Chamberlin, Morton M. Astrahan, Kapali P. Eswaran, Patricia P. Griffiths, Raymond A. Lorie, James W. Mehl, Phyllis Reisner, Bradford W. Wade, "SEQUEL 2: A unified approach to data definition, manipulation and control" *IBM Journal of Research of Development*, 20(6): pp. 560 - 575, 1976.

[6] M. M. Astrahan, D. D. Chamberlain, "Implementation of a Structured English Query Language", Communications of the ACM, 18 10, Oct. 1975 pp. 580-587.

[7] N. C. Shu, B. C. Housel, V. Y. Lum, „CONVERT- A High Level Translation Definition Language for Data Conversion“, *Communications of the ACM*, Vol.18, Nr.10,Oct., 1975, pp.. 557-567.

[8] K. Benecke, *"A powerful Tool for Object-Oriented Manipulation"*, in "Object-Oriented Databases: Analysis, Design & Construction (DS-4) R.A. Meersmann, W. Kent, S. Khosla (editors), Elsevier Science Publisher B.V. (North Holland) 1991, pp. 95-121.

[9] S. Abiteboul, N. Bidot, „*Non First Normal Form Relations: An Algebra Allowing Data Restructuring“*, Rapports de Recherche No347, Institute de Recherce en Informatique et en Automatique, Rocquencourt, France, Nov. 1984.

[10] W3C,“XQuery 3.1: An XML Query Language”, W3C Recommendation 21 March 2017, Status Update (6 April 2021), [Online]. https://www.w3.org/TR/xquery-31/ (Accessed Date: December 9, 2024).

[11] J. D. Ullman, *"Principles of Database Systems"*, Computer Science Press, Rockville, Maryland 1982.

[12] H. Zemanek, *"Formal Definition the Hard Way"*, Proc. IFIP TC2 Working Conference, Vienna 1985; North Holland, pp. 411-417.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.