o++o Tabments - Queries, Calculations, Statistics and Visualization

Klaus Benecke

(07.08.2025)

Foreword

What does it mean that o++o (ottoPS) is probably the simplest programming language?

It does not mean that o++o consists of very simple concepts. It does mean, however, that its application is relatively simple. o++o is not simple, but solving problems is easier than with other programming languages of equal expressiveness. o++o behaves like a natural language. English or German is not easy to learn either. However, natural language can be used - to a certain extent - even by children under the age of four.

We are convinced that the basic idea behind our best operation (stroke list operation) is easier to understand than the multiplication algorithm of decimal numbers. In our opinion, the concepts of o++o are relatively difficult to formalize, but they can often be described by simple algorithms that almost every user (= OttoNormalVerbraucher) can use in the future.

Is o++o a programming language?

o++oPS is designed as an end-user language, but not for programming complex database systems or compilers. It was developed to support people in solving their mathematical everyday requirements. Daily challenges are first of all (ad hoc) queries to tables (databases), documents or collections of tables and documents. It also includes financial calculations or in other daily context: determination of function values, determination of zeros or extrema of functions and solving a system of equations (calculation with matrices). In addition, o++o should be able to generate and manipulate images and visualize tables and documents in the form of diagrams. The most important innovative ideas of o++o compared to other approaches are connected with repeating groups. This means that a given object may contain not only null or one value for an attribute, but also multiple values. For such structures, known for more than 50 years in computer science, o++o provides new, powerful and easy to use operations.

This book contains a variety of sample queries to illustrate the basic concepts.

Table of content

List of programs and queries

Introduction

We summarize the main design principles and requirements for an end-user computer language or data model with corresponding operations:

1. It should be based on easily applicable concepts with a simple syntax.
2. It should be expressive and powerful.
3. It should be expandable with new operations.
4. It should have precise semantics based on algorithms.
5. It should allow queries on tables (databases) and documents.
6. It should allow queries over document collections (IR systems) and entire databases.
7. It should allow at user level computations by naive (brute force) algorithms.
8. It should also be usable for people with little interest in mathematics and computer science (programming by gut feeling).
9. It is intended to provide simple as well as more sophisticated concepts for broad classes of applications, suitable even for users with a keen interest in mathematics and computer science.
10. It should solidly integrate single data and mass data operations.
11. The result of a mass data operation should be as small as possible.
12. It would be nice if it could use graphical features based on structured tables.
13. It should be efficiently implementable.
14. At least parts of the language should be able to be optimized.

o++o was designed and developed with these principles in mind. It started as a database language for tables with repeating groups. A record with repeating groups may contain not only one value at each position, but also several (sub tuples of) values. For example, a student record may contain a name and a scholarship, but it may also contain multiple hobbies or multiple (SUBJECT,MARK) pairs. Similarly, a machine part may contain a number and a color, and to that several subparts or several layers or edges. Such sub-tuples may have sub-tuples again. These repeating groups have existed in computer science for more than 60 years. They are typical for hierarchical systems (IMS, ...), but were later discredited by emerging relational systems. Even today they are widely used in XML, JSON and NoSQL systems. However, in our opinion, there is no widely accepted computer language capable of adequately handling these richer structures. With the advent of XML, we have been able to generalize our operations to the new capabilities of arbitrary tagging and the alternate operator (|). Therefore, we are able to manipulate not only tables, but also documents. We have introduced the name tabment. A tabment can be understood as an abstract (syntax-independent) specification of an XML document. Step by step we improved our language o++o. We introduced binary search trees in tabments. Thus, we have achieved great efficiency gains for several operations.

Some indices can also be considered as tabments.

Our language o++o has been implemented in OCaml. Some basic keywords of o++o are German or Spanish ('gib' instead of SELECT, 'si' instead of true, ...) because they are shorter than the corresponding English words, but most keywords are English. This seems to be important because smartphones have only a small screen.

Today, many people also believe that no one would buy a computer program that might take a few hours or days to learn.

We put forward the following arguments against it:

0. Almost all people in the world had to learn for several years to understand the single data operations addition, multiplication, division and difference for each number range in school.

Are mass data operations like selections, calculations, restructuring, sorting tables, ... not just as important?

1. A good programming language allows many problems to be formulated more briefly and precisely with fewer misunderstandings than any natural language.
2. There is no need to explain the advantages of someone who has a driver's license or even a car. If he can even program solutions to problems with the computer himself, this increases the quality of computer use, because he can also interpret the results better. He does not need a computer scientist (chauffeur). Thus, there are fewer communication problems and he saves the cost of the computer scientist and the time for communication.
3. If the individual can make precise queries, he has much more compact query results and saves a lot of manual search effort. This also reduces the workload and improves quality.

What are the more specific design principles of o++o?

1. **important things first**

1.1 Sorting by the first attributes of a collection

```
gib DEPARTMENT,CHIEF,(NAME,LOCATION m) m
```

Here is described a structured table, which contains for each department also a corresponding group of employees. Sets (m) (and multi sets) are always sorted by the first column names. I.e. the outer set is sorted by DEPARTMENT and the inner set by NAME and then by LOCATION, because the NAME is not always a key in a department.

1.2 First written - first calculated:

`2+3*4` gives `20`

Here, a rectangle has one longer side, which consists of two sections 2 and 3 meters long. The other side is 4 meters long. The area gives 20.

`3*4+2` gives `14`

If I have two rectangles, one with side lengths 3 and 4 and one with area 2, I can first calculate 3*4 and then add 2 to get the area.

Please, do not overestimate these two examples. This is not new, because you get the same results with the most pocket calculators and the calculator of WINDOWS in normal mode. Also, the computer language SMALLTALK computes from left to right. Further, the old Greek scientists and each child in the first and second grade ,… compute in this way. Additionally, very few people know all priority rules.

1.3 TT-Invariance (TT=TabmentType)

For many operations such as addition or multiplication, the type of the result is the same as the type of the first input value.

```
<TABH!
SUBJECT, MARKl m
Math     1 2 4 1
Phy      2 3 5 2 4
!TABH>
*15/6
```

Here a whole table in horizontal tab format is multiplied by a number. That is, each number of the table is multiplied by 15/6 and the words remain unchanged. This results again is a table of the type SUBJECT,MARKl m. We see again, the first input value is more important than the second.

1.4 Exponent representation of numbers

o++o additionally, a representation allows, where the more important part - the exponent - precedes the mantissa of a number. The exponent says more about the size of the number than the mantissa:

6m12.345'678 (12 million 345 thousand ...)
9m123.456 (123 billion 456 million ...)

2. pragmatics and methodology first

We can also allow multi-line semantics for a single term. Then we could replace

(23+45+67) * (1111+2222+3333+4444)

through

```
23+45+67
*
1111+2222+3333+4444
```

This can be typed faster and is also clearer by dedicating a line to each pair of parentheses. In o++o this notation is further shortened to

```
23+45+67
* 1111+2222+3333+4444
```

This is not only done for methodological reasons (better readability), but for pragmatism. This notation does not waste the additional middle line. Compared to the first notation, you have to use a (larger) return key only once instead of 4 brackets. Further, each line gives a result.

3. short catchy keywords

Short programs require short keywords and short operation symbols or names. However, if the number of these symbols becomes too large, one must also allow full names for designations so that the user can remember them. For o++o holds, the more important a symbol is, i.e. the more frequently it is used, the shorter it is. This rule can be better implemented by allowing non-English keys as well.
Very short are + , * ,... l (list), m ... . This is certainly all right. We have also replaced many English terms by more memorable and shorter symbols:
sum: ++
product: **
average: ++:
count: ++1
...

Where we have found very short memorable known words in a language other than English, we substitute English terms with shorter ones from other languages if those words are known to many people:
true: si (Spanish Italian)
false: no
From the translation of SELECT-FROM-WHERE (gib-aus-mit) are the German words:
gib (select) for "give me"
and
aus (from)

4. **programs are processed from top to bottom and from left to right.**

Programs with loops or general recursion are expressive and powerful, but often difficult to read and understand. Sequential programs are expected to be not so expressive. o++o has been developed to prove the opposite, too. This requires powerful and expressive operations.

Readability of programs and tabments is an important problem.
o++o has been taken this into account as follows:

1. Programs can often be written short.

2. Numbers can also be displayed in Swiss style (e.g.: 12'345'678)
3. Lines indented by more than 4 spaces logically belong to the previous line. E.G.:
   ```
   my_marks.tab
   gib AVG,(SUBJECT,AVG m)
       AVG:=MARK! ++:   # this line belongs from the logical
                  # point of view still to the previous line
   rnd 1
   ```
4. A structured table with the scheme
   ```
   DEPARTMENT,CHIEF,(NAME,SALARY m) m
   ```
   contains each department and boss only once. This not only reduces redundancy, but also improves readability compared to flat tables of this type.

In the chapters it is shown, how general and simple the query possibilities of o++o are. Chapter 1 introduces some basic functions of our "pocket calculator". All examples there do not require any stored tables or documents. This does not mean that our o++o programs cannot work with files. Tables and documents can also be stored.

First of all, the user must understand what a schema is and what are the tables or documents that belong to this schema. Then it will not be too difficult to grasp the query examples for selection, calculation and restructuring of the first chapters. All operations allow a compact and readable formulation of (complex) queries. They apply to nested lists or sets, and they are new to the database world. Calculations can often be understood as hierarchical "map" functions, because operations are often applied to each of the input values. Restructuring with the gib clause is very expressive, as it is combined with sort (m, b), duplicate elimination (m), aggregation (++, min, max, +
+1, ++:, or2, &&, **, variance).

We know of no other restructuring operation in a commercial product that allows to transform a given hierarchy only by specifying a schema or TT (Tabment Type) of the desired structure. Although the operations in the examples are only applied sequentially, they cover a wide range of applications.

Section 10.4 introduces a "natural" join called *ext* operation and its un-nested and nested uses. It becomes clear that we do not need the Cartesian product and even the ordinary flat relational join. A simplified notion of recursion is introduced in Chapter 3. With this end-user recursion, appropriate queries can be realized with minimal learning effort. After showing in Chapter 4 that printing two words is not just a syntactic issue, Chapter 6 tries to make clear that o++o is useful for all subjects in school, but especially for mathematics and computer science. It will be made clear that even 9th or 10th grade students can solve problems that are applications of differential and integral calculus. In addition, it is argued that the ordinary division algorithm could be eliminated from the mathematics curriculum. It requires neither Cartesian product nor (hidden) join conditions.

Chapter 17 contains some queries where the result can be interpreted as an image. Roughly speaking, each result table contains the coordinates of points possibly combined with a color value. It is also shown that it is easier to create structured diagrams based on structured tables.

The most important operations of the data model are described in more detail in chapter 10. Section 10.3 contains the description of the restructuring operation, 10.2 the assignment operation and 10.1 the selection.

Acknowledgements:
I would like to thank the following computer scientists for their valuable contributions to our system o++o and previous systems:

Wolfgang Reichstein for the first one-step implementation of the restructuring operation in C for HSQ files, Dmitri Shamshurko for the first implementation of the first core of the "gib-aus-mit" construct in a functional style (Caml Light),

Martin Schnabel for the conception and implementation of subroutines and other features,

Andreas Hauptmann for improving many concepts in design and efficiency, especially for query optimization concepts.

Further thanks go to Stephan Schenkl and Mirko Otto for supporting the o++o project.

1 Calculations and spreadsheet applications with o++o

We first present some numerical calculations.

Program 1.1: Addition;	Result (type: the more general type of both)
`1 + 4.56`	`5.56`

Program 1.2: Division	Result
`1:7`	`0.142857142857`

Program 1.3: Division with improved readability	Result
`1:7 '3`	`0.142'857'142'857`

Program 1.4: Division with rounding	Result
`1:7 rnd 3`	`0.143`

Program 1.5: Exponentiation	Result
`3^20 '3  # '3 is an idea from Swiss`	`3'486'784'401`

# is the comment character. Comments can be used to explain programs.

Program 1.6: Addition of rational numbers	Result
`3/4 + 1/3`	`13/12`

Program 1.7: Type of the first input value is maintained, if the types do not generalize each other	Result
`3/4 + 0.3`	`21/20`

Program 1.8: Type of the first input value is maintained, if the types do not generalize each other	Result
`0.3+3/4`	`1.05`

Program 1.9: Difference or list	Result

`3 - 2 # Note that "3 -2" is a`
`      # List of two numbers`
	`1`

Program 1.10: Sine of pi : 2	Result
`pi : 2 sin`	`1.`

Program 1.11: Sine of 30 degrees	Result
`30:180*pi sin`	`0.5`

Program 1.12: How many 10-digit binary	Result

numbers are there?	
`2 ^ 10      # base:2   exponent: 10`	`1024`

Program 1.13: Calculate the edge length of a cube with volume 2	Result
`2 ^ 1/3`	`1.25992104989`

or

Program 1.14: Calculate the edge length of a cube with volume 2 using ordinary division operation	Result
`2 ^ (1:3)`	`1.25992104989`

Program 1.15: Sum of 4 numbers	Result
`3.21 4.56 6.88 9.32 ++`	`23.97`

Program 1.16: Sum of numbers from 1 to 100	Result
`1 .. 100 ++`	`5050`

Program 1.17: Product of the numbers from 10 to 40	Result
`10 .. 40 **`	`2248443792019118536005322061276774400000000`

You can see from the result that you can process arbitrarily large integers with o++o.

Program 1.18: Maximum of numbers	Result
`1/3 2/7 max`	`1/3`

Program 1.19: Average of several marks	Result
`1 3 2 1 3 4 ++:`	`2.33333333333`

Program 1.20: Introduction of two column names (Output values of two terms simultaneously)	Result

`X:=2 ^ 10   # := : assignment`
`Y:=X : 10`
	`X,   Y`
	`1024 102.4`

Program 1.21: a pair of two independent terms	Result

`2 sqrt; 3 sqrt # ; separates`
`               # stronger than ,`
	`PZAHL,        PZAHL`
	`1.41421356237 1.73205080757`

There are few commas in primary data of tables. This would destroy the readability. Therefore, we do not find commas in .tab files, for example, even if pairs or tuples are represented. However, pairing is represented in the metadata (table headers) of the tables to prevent misunderstandings. PZAHL is a number with a point.

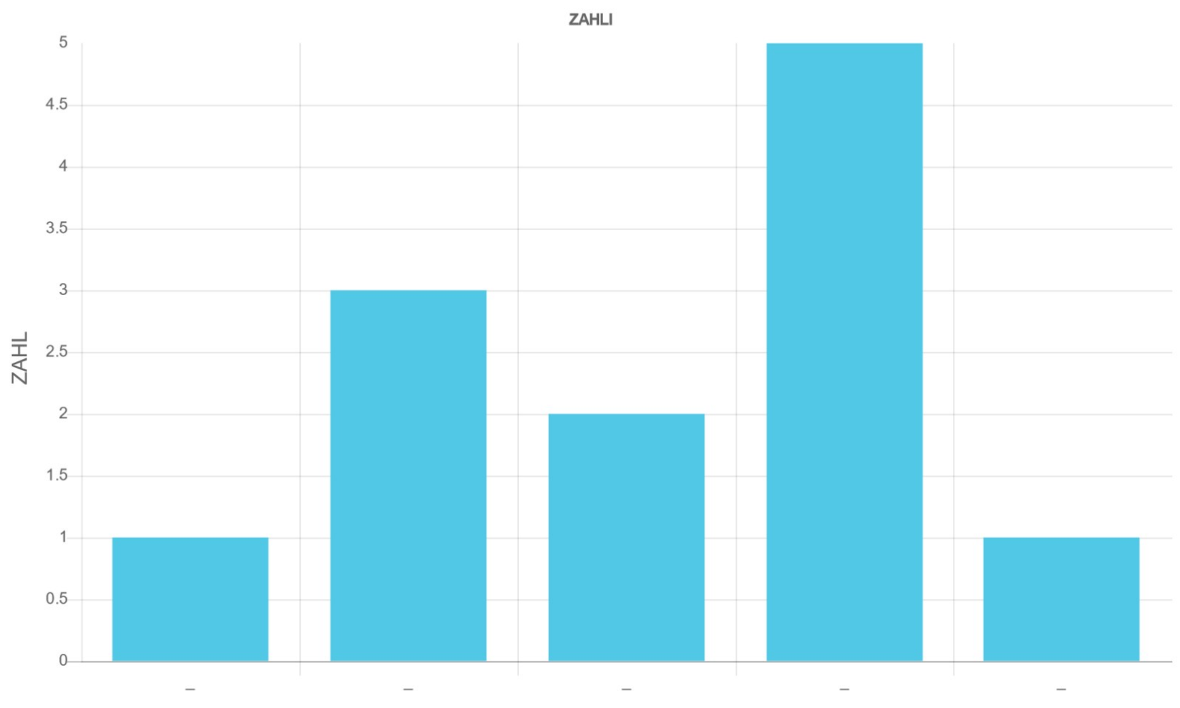
Program 1.22: Comma is an ordinary operation: Calculation from left to right	Result

`2 sqrt,3 sqrt  # the last sqrt`
`               # acts both via`
`               # "2 sqrt" as well`
	`PZAHL,      PZAHL`
	`1.189207115 1.73205080757`

`# as over 3`	

Program 1.23: divrest generates a pair of numbers	Result

`DIV,REST:=13 divrest 5`	`DIV, REST`
	`2    3`

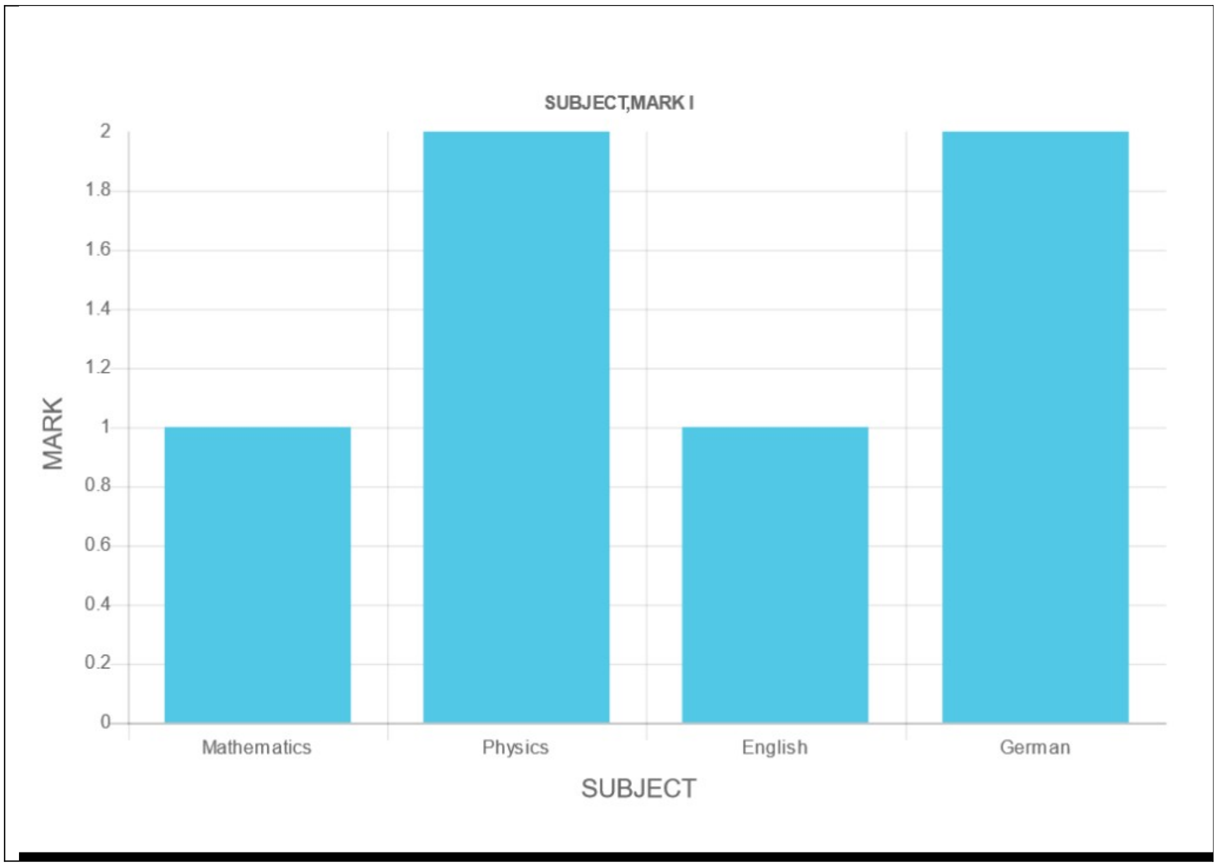
Program 1.24: create a simple diagram with one click
`1 3 2 5 1 # List of numbers`
Result: Diagram (columns) 

```
SUBJECT,    MARK l
Mathematics 1
Physics     2
English     1
German      2
```

marks1.tab

The above table represents a list of (SUBJECT,MARK) pairs. It can be created with any text editor or typed into the output field of the o++o interface. `l` stands for list.

Program 1.25: a simple bar chart with signatures
`marks1.tab`
Result (struc.diagram - bar)



It is also possible to enter the following line into the program field of the Otto interface.

```
SUBJECT,MARK l:=Mathematics Physics English German posjoin 1 2 1 2
```

By the operation ,, the both given lists are elementwise connected by comma. The resulting list consists of 4 (WORT,ZAHL) pairs, where the first column is renamed to SUBJECT and the second to MARK.

The basic data of the following query can be generated by the following small structured table. Here `l` stands for list. It needs the ending ".tabh", because the marks are arranged horizontally. Lists were invented in Venice (Lista). The single entries (=elements = rows) of the list were arranged one below the other. The subjects are also arranged vertically in noten2.tabh. Simple lists were already arranged horizontally thousands of years ago. A sentence is a list of words, which were essentially arranged horizontally. Since this saves a lot of screen space and paper, simple (single-column) lists in o++o can also be arranged horizontally. This is possible because the list is understood abstractly. This allows o++o to understand JSON lists, for example, even though the list elements are not simply separated by spaces. In questions of the representation of the elements, sets and multisets are equal to lists. However, different parentheses are used.

SUBJECT,	MARKl m
Mathematics	2 1 3
Physics	2 2 3
English	1 4

marks2.tabh

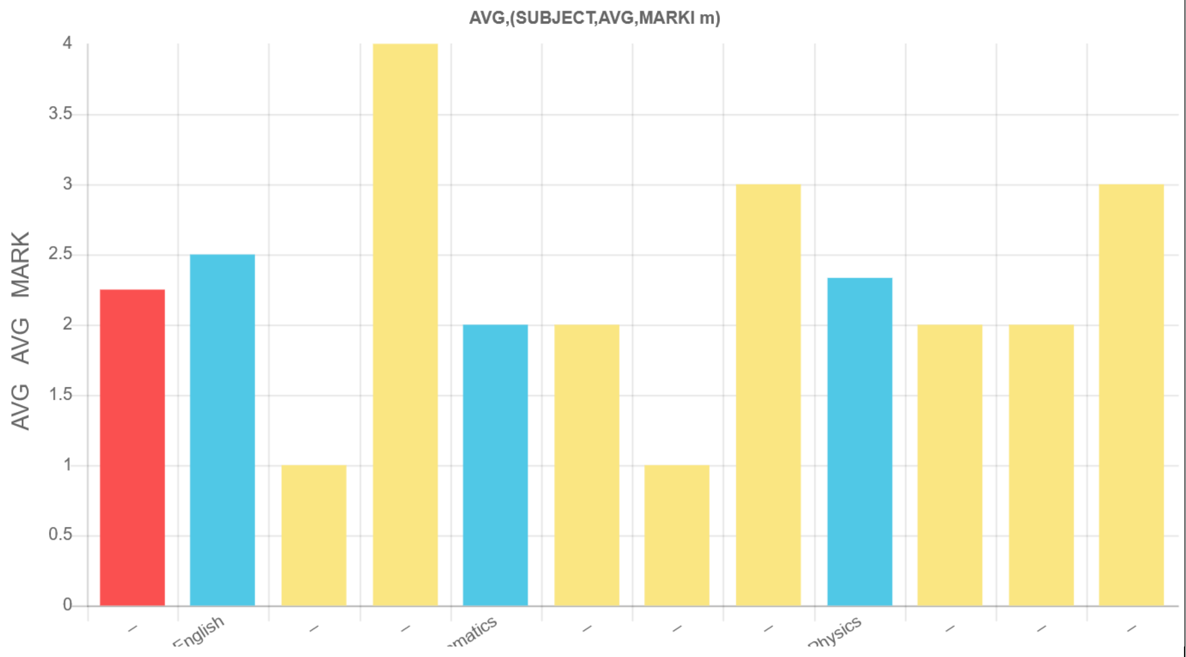
This table can also be generated by the following program line with set brackets { }:

```
SUBJECT,MARKl m:={Mathematics,[2 1 3] Physics,[2 2 3] English,[1 4]}
```

Program 1.26: a structured diagram

```
marks2.tabh
gib AVG,(SUBJECT,AVG,MARKl m)
    AVG:=MARK! ++:
```

Result (diagram columns)



Result (tabh output)

```
AVG, (SUBJECT,    AVG,         MARKl m)
2.25  English     2.5          1 4
      Mathematics 2.           2 1 3
      Physics     2.333333333 2 2 3
```

The following are examples of a curve discussion using a parabola as an example.

Program 1.27: Calculation of a small table of values of the quadratic function with coefficients 1 -8 13 (x^2 - 8 x +13)

```
Xl:= -2 .. 10
Y := X poly [1 -8 13]
```

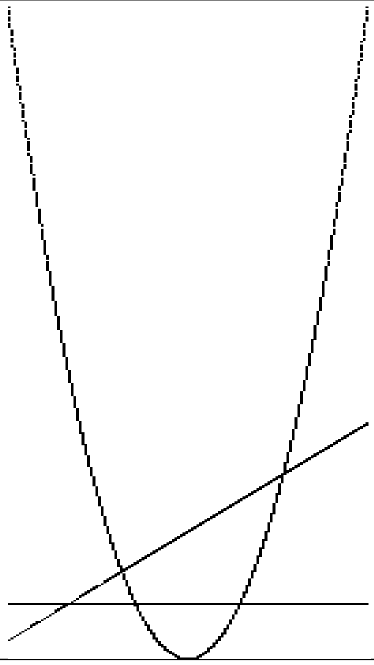
Result (tab)

```
X, Y l
-2 33
-1 22
 0 13
 1 6
 2 1
 3 -2
 4 -3
 5 -2
 6 1
 7 6
 8 13
 9 22
10 33
```

Program 1.28: Expanding the value table so that a function graph can be seen.
Draw the graph of the parabola (quadratic function) with the x-axis and the function y=x in the interval [-2 10].

```
Xl:= -2 .. 10! 0.01
Y := X poly 1 -8 13
LINE:= X
Y0:= 0*X
```

Result (image)



Program 1.29: Approximate determination of the (local) minimum of the parabola

```
-2 ... 10!0.0001 poly [1 -8 13] min
```

Result

```
-3
```

Program 1.30: Approximate determination of the two zeros

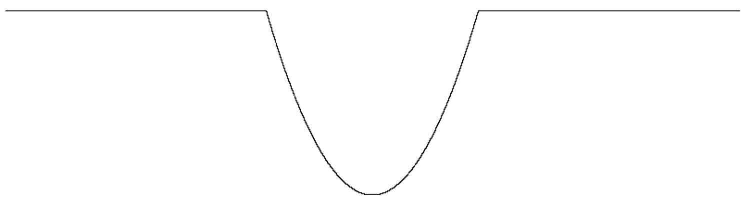
```
Xl:= -2 ... 10!0.0001
Y := X poly [1 -8 13]
sel  Y succ * Y <= 0 # succ: successor
rnd 7
```

Result (tab)

```
X,         Y l
2.2679000  0.0001704
5.7320000 -0.0001760
```

Program 1.31: Determining the area under a (composite) function

Result (image) (without 2 last program lines)



```
Xl:= -2 ... 10! 0.0001
Y := (X poly [1 -8 13],0) min
RECTANGLE:= Y*0.0001
++ RECTANGLE
```

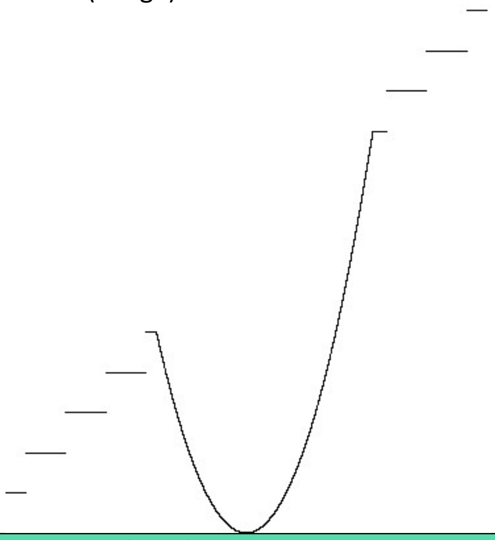
Result (tab)

```
-6.92820323316
```

If we omit the last two program lines in the following program, the function can be visualized by clicking on *image*:

Program 1.32: Determination of the area under a non-continuous function

Result (image)



```
Xl:= -2 ... 10! 0.0001
Y := X poly (1 -8 13),(X rnd 0) min
RECTANGLE:=Y*0.0001
++ RECTANGLE
```

Result (tab)

```
21.970089131
```

Program 1.33: Using the divrest function to output number pairs

```
Xl:=1 ..10
DIV,REST:=X divrest 3
```

Result (tab)

```
X, DIV,REST l
  1 0   1
  2 0   2
  3 1   0
  4 1   1
  5 1   2
  6 2   0
  7 2   1
  8 2   2
  9 3   0
 10 3   1
```

Program 1.34: Determination of all prime numbers up to 70

```
Xl:= 2 .. 35
Yl:= 2 .. 9 at X
PRODUCT:= X*Y
sel  PRODUCT <= 70 # sel : with
gib PRODUCTm
PRIMl:= 2 ..70
sel- PRIM in PRODUCTm   # sel-: without
gib PRIMl
```

Result (tabh output):

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67
```

Program 1.35: Calculate the circumference of several circles, whose radii are given. The results are to be rounded to 2 digits after the point.

```
4 5 6 2 3.7 9.77 *pi*2 rnd 2
```

Result (tabh)

```
25.13 31.42 37.7 12.57 23.25 61.39
```

You can see that this program can be written in one line.

Program 1.36: Calculating circumference and area of several circles, whose radii are given

```
Rl:= 4 5 6 2 3.7 9.77
CIRCUM :=R*pi*2
AREA:=R*R*pi
rnd 1
```

Result (tab)

```
R,  CIRCUM, AREA l
4.   25.1     50.3
5.   31.4     78.5
6.   37.7    113.1
2.   12.6     12.6
3.7  23.2     43.0
9.8  61.4    299.9
```

By Rl:= the name R (called "tag") is assigned to each element of the given list.

An assignment (":=") adds a new column to the specified table. In the above program, the columns CIRCUM and AREA are added one after the other, resulting in a table of type R,CIRCUM,AREA l. l stands for list. Unfortunately, this can easily be confused with the one.

Program 1.37: Calculating perimeter and area of multiple rectangles

```
<TAB!
A,   B l
1.23 5.67
7.65 4.32
9.87 6.54
!TAB>
CIRCUM:=A+B*2
AREA:=A*B
```

Result (tab)

```
A,   B,   CIRCUM, AREA l
1.23 5.67 13.8      6.9741
7.65 4.32 23.94    33.048
9.87 6.54 32.82    64.5498
```

The TAB brackets ("<TAB!", "!TAB>") are needed only in the program part of the system. In a file the system recognizes the type by the ending ".tab". In the TAB representation the values must be aligned to the left side of the associated column names.

Program 1.38: Total price of a simple invoice

```
<TAB!
ARTICLE, PRICE l
Beer     0.61
Lemonade 0.23
Steak    2.40
!TAB>
++
```

Result

```
3.24
```

Here we simply sum over the numbers in the given table (a list of pairs). The ARTICLE values are words and therefore have no effect on the result. Now we replace ++ with +% 10. This creates a table with 2 columns and three rows (records, tuples). Each number now still contains 10% tip:

Program 1.39: Multiplying a table by a number

```
<TAB!
ARTICLE, PRICE l
Beer     0.61
Lemonade 0.23
```

Result

```
ARTICLE, PRICE l
Beer     0.671
Lemonade 0.253
Steak    2.64
Steak     2.40
!TAB>
+% 10
```

Then you can add again with ++ to get the total (3.564).

Program 1.40: Find the total price of a more complicated calculation using a simple table

```
<TAB!
ARTICLE, PRICE, CNT l
Beer      0.61   7
Lemonade 0.23   3
Steak     2.40   4
!TAB>
POSPRICE:= PRICE*CNT
gib POSPRICEl
++
```

Result

```
14.56
```

As a result of the assignment, the specified table is extended by a new column with the column name POSPRICE, where each of the three PRICE values is multiplied by the associated CNT value. To determine the total price the ++ operation has to be applied only to the POSPRICE-values. Otherwise, the sum of all nine numbers in the table above would be formed.

The first input value of an operation, which is at the beginning of a program line, is always the result of the previous program line.

The ":=" sign of the assignment is to be distinguished from the equal sign =. For the formulation of conditions the equal sign, as well as <, >, <=, "in" etc. is needed. Conditions are used for selection (filtering of (complex) rows of structured tables).

For example, add a condition
`sel  ARTICLE = beer`
or only
`sel  beer`
then the final result is the total price for the seven beers. If you want to calculate only the price for the other items instead, use
`sel- ARTICLE = beer`
or simply
`sel- beer`.

Column names (metadata) must always be written in upper case. The keywords (gib, sel-, sel , ...) must always be written in lower case. If you write a word of the primary data always with upper and lower case letters, the program becomes easier to read.
The reference to the aggregation (here ++) results from the header line of the desired table. TOTAL is an aggregation per NAME. Sets (m, m-) are always sorted by the column names specified first.

Program 1.41: Find a weighted average for 3 students and the overall average

```
<TABH!
NAME,  EXAMl,MARKl l
Ernst  1 2   1 2 3 1 3 1 1
Clara  1 1   3
Sophia 1 3   1
!TABH>
TOT:=EXAMl ++: *0.6 +(MARKl ++: *0.4)
```

Result (tabh)

```
TOTAL,(NAME,  TOT, EXAMl,MARKl l)
1.66   Ernst  1.59 1 2   1 2 3 1 3 1 1
       Clara  1.80 1 1   3
       Sophia 1.60 1 3   1
```

`TOTAL:=TOTl ++:`
`rnd 2`	

Program 1.42: A woman weighs 40 kg plus half her weight. How much does she weigh?	Result

`WEIGHTl:= 40 .. 100`
`sel  WEIGHT:2+40=WEIGHT`
	`80`

Program 1.43: A bottle with a cork costs one euro and ten cents. The bottle is one euro more expensive than the cork. How much does the bottle cost?	Result

`BOTTLEl := 0 .. 110`
`sel  110-BOTTLE=BOTTLE- 100 # = CORK`
	`BOTTLE`
	`105`

Program 1.44: A bottle with a cork costs one euro and ten cents. The bottle is one euro more expensive than the cork. How much does the cork cost?	Result

`BOTTLEl:= 0 .. 110`
`CORK:= BOTTLE - 100`
`sel  CORK+BOTTLE = 110`
	`BOTTLE, CORK l`
	`105      5`

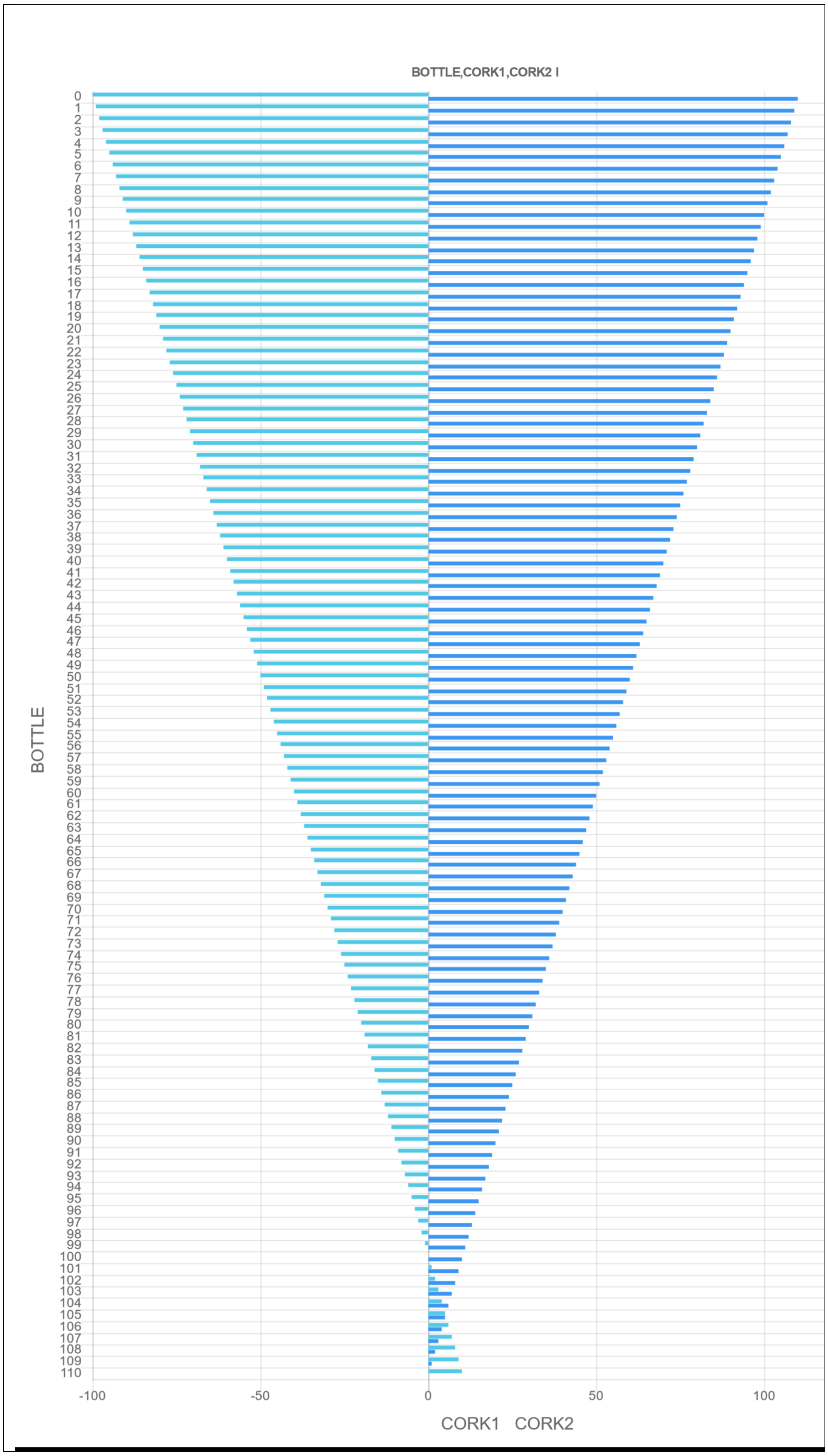
The first assignment gives each of the numbers from 0 to 110 the tag BOTTLE. This is best seen by looking at the *ment* representation:	If we had written the assignment BOTTLE:= 0 ..110, the BOTTLE tag would appear only once:

`<TABM>`
`  <BOTTLE>0</BOTTLE>`
`  <BOTTLE>1</BOTTLE>`
`  <BOTTLE>2</BOTTLE>`
`  <BOTTLE>3</BOTTLE>`
`  ...`
`  <BOTTLE>108</BOTTLE>`
`  <BOTTLE>109</BOTTLE>`
`  <BOTTLE>110</BOTTLE>`
`</TABM>`
	`<BOTTLE>`
	`0`
	`1`
	`2`
	`3`
	`...`
	`108`
	`109`
	`110`
	`</BO TTLE>`

Program 1.45: A bottle with a cork costs one euro and ten cents. The bottle is one euro more expensive than the cork. How much does the bottle cost?	Result

`BOTTLEl:= 0 ..110`
`CORK1   := BOTTLE - 100`
`CORK2   := 110 - BOTTLE`
`sel  CORK1=CORK2`
	`BOTTLE, CORK1, CORK2 l`
	`105      5      5`

This solution is advantageous from a methodical point of view, because the first 3 program lines can be displayed by clicking on the image button. You can see that there are 2 straight lines whose intersection is determined by the conditions. You can also click diagram/Balken to get the following result, where it is visible that both bars are equal at 105. Here we had to add the program line: BOTTLE::=BOTTLE wort


BOTTLE,CORK1,CORK2 I
BOTTLE
CORK1 CORK2
-100
-50
0
50
100

2 A savings bank account

The following requests refer to data records of the savings bank. Here the customer can download his data as a csv file. csv files have a very simple structure. Since they contain a lot of quotation marks, they are relatively difficult to read. The otto user does not need to familiarize himself with this syntax. He can view them or parts of the file in the usual way as *tab, hsq, ment, web* or *json* files. We consider a file `turnover.csv,` which contains transactions from 3 years.

Program 2.1: How many turnovers are there?

```
turnover.csv
++1
```

Result

```
162
```

Program 2.2: Give the first columns of the first and last transaction!

```
turnover.csv
sel  AMOUNT pos=1 or AMOUNT pos- =1
```

Result (tab output):

```
ORDERACCOUNT,     POSTINGDATE,VALUEDATE,POSTINGTEXT,    USAGE....
DE598105327206411 20.07.22    20.07.22  ONLINE REFERRAL ReNr2
DE598105327206411 13.05.20    13.05.20  ONLINE REFERRAL Wage
```

pos determines the position of a tuple. pos starts counting from the beginning with 1 and pos- from the end. "or" is the logical or sign.

Program 2.3: How much money was transferred to the account?

```
turnover.csv
sel AMOUNT > 0
gib AMOUNTl
++
'3
```

Result

```
110'729.17
```

Program 2.4: How much money was transferred from the account?

```
turnover.csv
sel AMOUNT < 0
gib AMOUNTl
'3
```

Result

```
-94'713.65
```

Program 2.5: Contrast total revenues and expenditures

```
turnover.csv
gib INCOME,EXPENSES
    INCOME  := AMOUNT if AMOUNT > 0!0!++
    EXPENSES:= AMOUNT if AMOUNT < 0!0!++
'3
```

Result (tab)

```
INCOME ,    EXPENSES
110'729.17  -94'713.65
```

Program 2.6: How much was transferred to Ms. Heyer in total?

```
turnover.csv
sel Heyer
```

Result

```
-54'538.28
```

```
gib AMOUNTl
++
'3
```

Here the user must know his data. If there are two Heyer's, the above result is certainly not the desired one. One could then add the first name:

sel Heyer Erika

or

sel Heyer & Erika

You can also use the account number, but then the program is not so well readable, because most people cannot remember an account number or IBAN.

Program 2.7: How much was transferred to Ms. Heyer in each year and month?

```
turnover.csv
sel Heyer
D,MONTH,YEAR:=VALUTADATUM zahltrip
gib YEAR,SUM,(MONTH,SUM m) m
    SUM:=AMOUNT ! ++
'3

rnd 2
```

Result (tab)

```
YEAR, SUM ,      (MONTH, SUM2 m)m
20    -16'807.22  5      -1'967.66
                  6      -1'777.31
                  7      -1'943.89
                  8      -2'110.27
                  9      -1'833.69
                 10      -3'189.39
                 11      -1'973.48
                 12      -2'011.53
21    -20'727.13  1      -2'011.53
                  2      -1'911.90
                  3      -2'006.17
                  4      -2'443.65
                  5      -2'438.02
                  6        -583.32
                  8        -812.47
                  9      -2'636.73
                 10        -993.13
                 11      -2'635.68
                 12      -2'254.53
22    -17'003.93  1      -1'630.30
                  2      -2'202.03
                  3      -1'324.38
                  4      -1'438.72
                  5      -4'460.36
                  6      -2'999.59
                  7      -2'948.55
```

Program 2.8: Give a comparison of the income and expenses for each year!

```
turnover.csv
YEAR:=20 wort + (VALUTADATUM subtext 7!2)
gib YEAR,PLUS,MINU,SUM m
    PLUS:=AMOUNT if AMOUNT>0!0!++
    MINU:=AMOUNT if AMOUNT<0!0!++
    SUM:=AMOUNT!++
'3
```

Result (tab output):

```
YEAR, PLUS ,    MINU ,     SUM m
2020  24'921.   -23'257.11  1'663.89
2021  50'468.04 -34'274.22 16'193.82
2022  35'340.13 -37'182.32 -1'842.19
```

subtext needs 3 input values. Here VALUTADATUM is the first, the initial character number 7 the second and the length of the desired string 2 the third. VALUTADATUM is here a word constructed in German date notation, e.g.: 18.06.21. In the above example, however, the year is to be output with 4 digits. For this, the word "20" must be concatenated with the two digits that subtext determines.

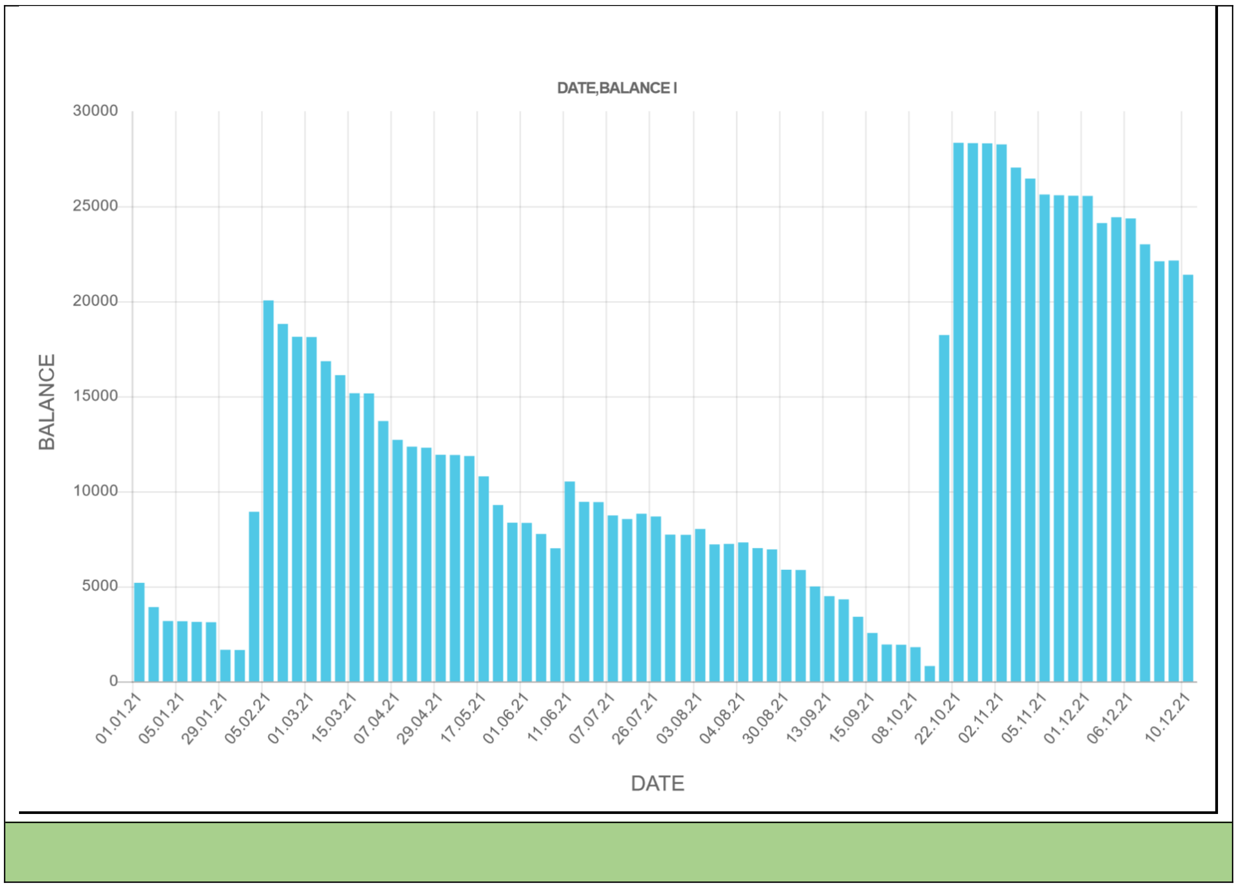
Program 2.9: Give me for each month of the year 2021 the income and expenses with the larger transfers!

```
turnover.csv
sel VALUTADATUM subtext 7!2 = 21
MONTH:=VALUTADATUM nthzahl 2
USE:=VERWENDUNGSZWECK subtext 3!6
RECIPIENT:=BEGUENSTIGTER_ZAHLUNGSPFLICHTIGER subtext 3!6
gib MONTH,PLUS,MINU,(AMOUNT,USE,RECIPIENT b-) m
    PLUS:=AMOUNT if AMOUNT>0 ! 0 ! ++
    MINU:=AMOUNT if AMOUNT<0 ! 0 ! ++
'3
sel  AMOUNT! AMOUNT abs>2'000
rnd 2
```

Program 2.10: Output the account balances of 2021 as a bar chart output!

```
turnover.csv
rename VALUTADATUM!DATE
sel DATE subtext 7!2=21
gib DATE,BETRAG l-
BALANCE:=  5'200 +BETRAG next BALANCE pred +BETRAG at BETRAG
gib DATE,BALANCE l
```

Result (bar chart):



Here, it is assumed that the initial account balance is 5'200. This number is simply added to the first AMOUNT (BETRAG) value in the above example.

The query possibilities of an account file and other files depend on the existing data. If, for example, no name for the recipient is given in the purpose of use of the data records, it is not possible to write well readable queries. The better the data material, the simpler the o++o programs and the more queries are possible. But this also makes clear that the one who knows the input data can write the best o++o programs. A computer scientist, who wants to program general evaluations of such data, will never be able to make the variety possible, which an end-user reaches, who knows the contents of the records exactly. The intended use alone offers many possibilities to improve the evaluations, which are certainly not yet exhausted by many.

The importance of a simple query language will be magnified when money transactions in Germany are also completely cashless. If everyone has access to the data on their purchases at a supermarket or gas station, they will be able to determine exactly when and for what they spent their money.

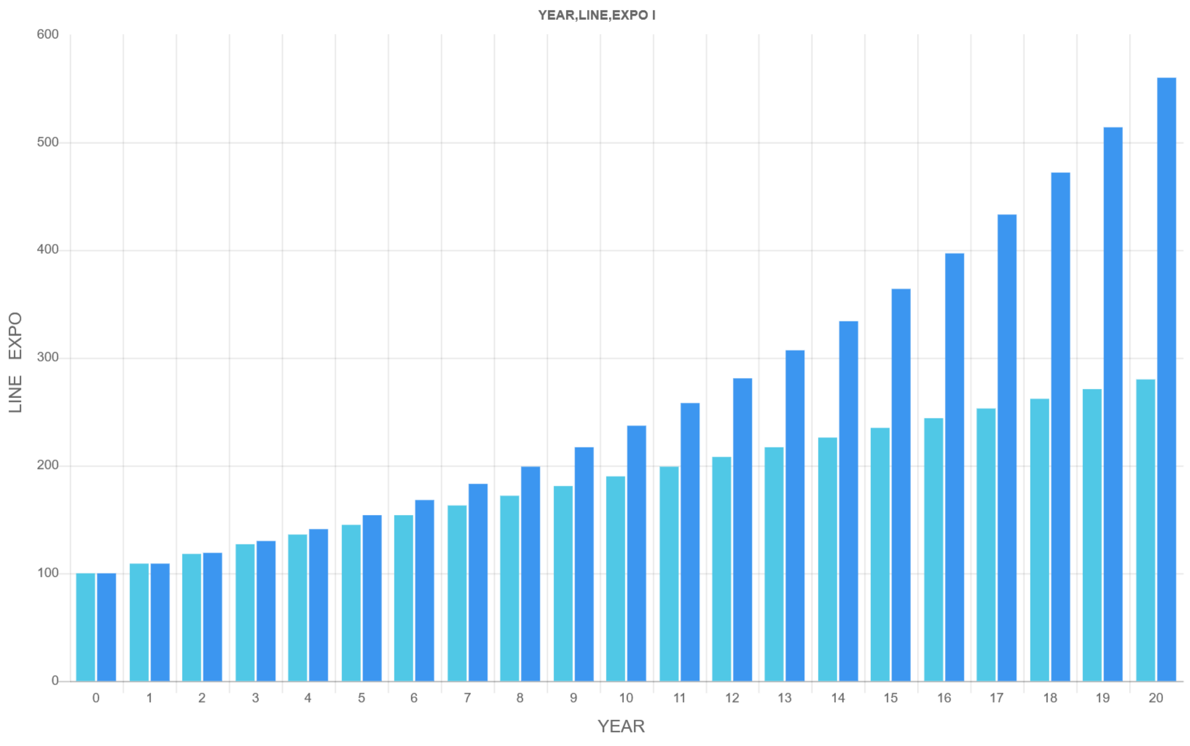
3 Table Recursion - Exponential Growth

Recursion is a powerful tool to describe functions or data structures in a short form. It is especially used in functional languages like OCaml or HASKEL. We introduce a type of "forward recursion" that is easy to use. An initial value is always described by a value or a term and the following values result from the direct predecessor by means of a second term, respectively. All generated values are visible in the result table.

Program 3.1: Compare linear and exponential growth within 20 years.

```
YEARl := 0 ..20
LINE  := 9*YEAR +100
EXPO  := 100. next EXPO pred +% 9 at LINE
rnd 0
YEAR  ::= YEAR text
```

```
Result (bar chart):
```



```
Result (tab):
YEAR, LINE, EXPO l
0     100   100.
1     109   109.
2     118   119.
3     127   130.
4     136   141.
5     145   154.
6     154   168.
7     163   183.
8     172   199.
9     181   217.
10    190   237.
11    199   258.
12    208   281.
13    217   307.
14    226   334.
15    235   364.
16    244   397.
```

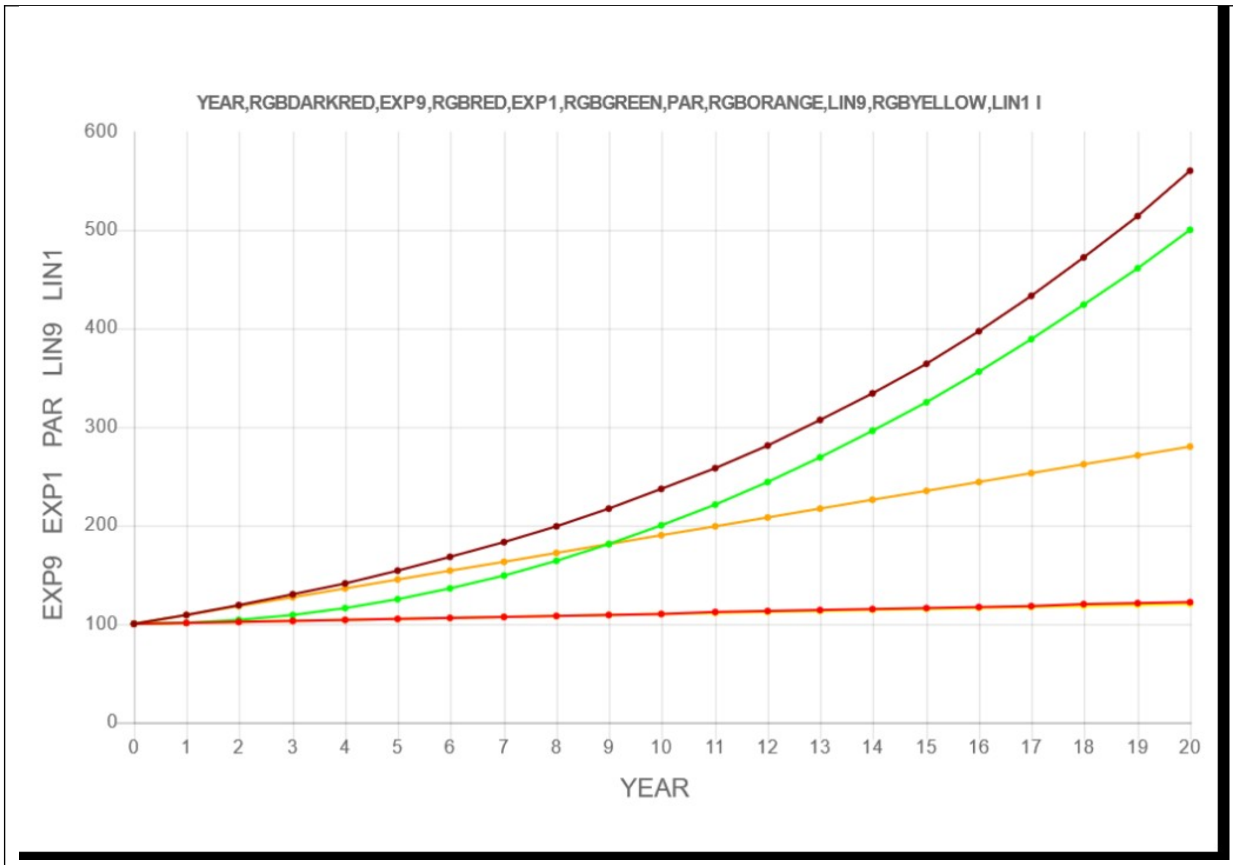
```
17    253   433.
18    262   472.
19    271   514.
20    280   560.
```

In the following program way one can describe exponential growth. EXP9 and EXP1 are the program lines for this. At the same time, these two columns in the tab representation contain the growth values. One percent growth is exponential growth if compound interests are taken into account. This is given in both formulas. If one adds in each case only 9% or 1% of 100 to the predecessor, then the interest of the interest is not considered. This would be the growth if one takes the interest from the interest every year. Our formulas for LIN9 and LIN1 correspond to this. These straight-line formulas and the exponential function curves differ here only at one point. If the operation + is replaced by + %, linear growth becomes exponential.
As is well known, exponential growth is far superior to any other growth and thus especially to linear growth. The fact that nine percent interest yields a far better total amount after 20 years than one percent is shown by the last line of the table (€560 versus €122). Without compound interest, the results are €280 and €120, respectively. To improve the comparison with polynomial growth, we have included a parabola.
The green parabola obviously shows a similar behavior in this range of 20 years as the exponential growth of 9 percent (dark red). In the next example we will see that this changes completely if we look at 200 years instead of 20. The yellow curve (1 % without compound interest) and the red curve (exponential growth 1 %) practically did not differ at all.
It should already be mentioned here that the curves are "distorted" so that they look nicer. In school, LIN1 would have to be drawn with an angle of 45°. LIN9 would be almost vertical with an angle of more than 83°. If you did that, the values of the fast-growing functions would have no place on the paper or screen, or you would have to shorten the x-axis (here YEAR) accordingly. But then it would look as if all points and curves were vertical. This undistorted real representation of the points is realized by the output 'image'. This doesn't look nice, but people should be confronted with reality from time to time. Then they can also better classify the visualizations below.

Program 3.2: How does an amount of 100 Euro develop with a "simple" and normal interest rate of 1% and 9%? and with quadratic growth within 20 years.

```
YEAR1:= 0 .. 20
EXP9,EXP1 :=  100.,100. next  preds +% (9,1) at YEAR
PAR   := YEAR * YEAR + 100
LIN9,LIN1 :=  100.,100. next  preds +  (9,1) at PAR
rnd 0
YEAR::=YEAR text
RGBDARKRED :=darkred leftat EXP9
RGBRED     :=red     leftat EXP1
RGBGREEN   :=green   leftat PAR
RGBORANGE  :=orange  leftat LIN9
RGBYELLOW  :=yellow  leftat LIN1
```

Result (line graph):



Result (tab)

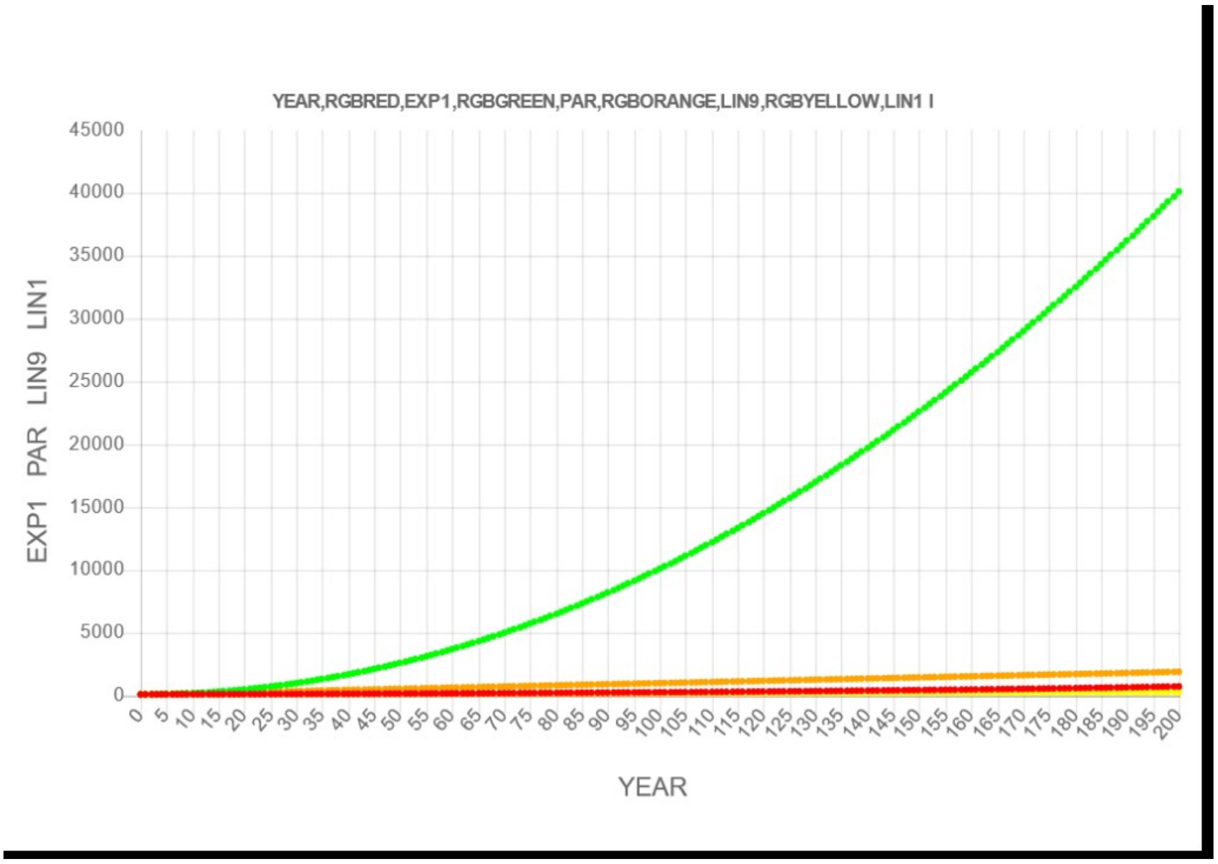
```
YEAR,EXP9,EXP1,PAR,LIN9,LIN1 l
   0 100. 100. 100 100. 100.
   1 109. 101. 101 109. 101.
   2 119. 102. 104 118. 102.
   3 130. 103. 109 127. 103.
   4 141. 104. 116 136. 104.
   5 154. 105. 125 145. 105.
   6 168. 106. 136 154. 106.
   7 183. 107. 149 163. 107.
   8 199. 108. 164 172. 108.
   9 217. 109. 181 181. 109.
  10 237. 110. 200 190. 110.
  11 258. 112. 221 199. 111.
  12 281. 113. 244 208. 112.
  13 307. 114. 269 217. 113.
  14 334. 115. 296 226. 114.
  15 364. 116. 325 235. 115.
  16 397. 117. 356 244. 116.
  17 433. 118. 389 253. 117.
  18 472. 120. 424 262. 118.
  19 514. 121. 461 271. 119.
  20 560. 122. 500 280. 120.
```

The new columns EXP9, EXP1 are defined by two formulas. The first element of the list of years is assigned the value of the first formula. The second value is calculated by the second formula, where "EXP9 pred" is the value of the predecessor and preds are both predecessors. Therefore, we get 100. +% 9=109 for the second value of the EXP9 column. The third value is again calculated by the second formula, but now we have to calculate 109 +% 9=118.81 (rounded to 119 at the end of calculations). In the same way, all the following values are calculated value by value using the second formula. The rounding does not cause any inaccuracies, because it is done after all calculations.

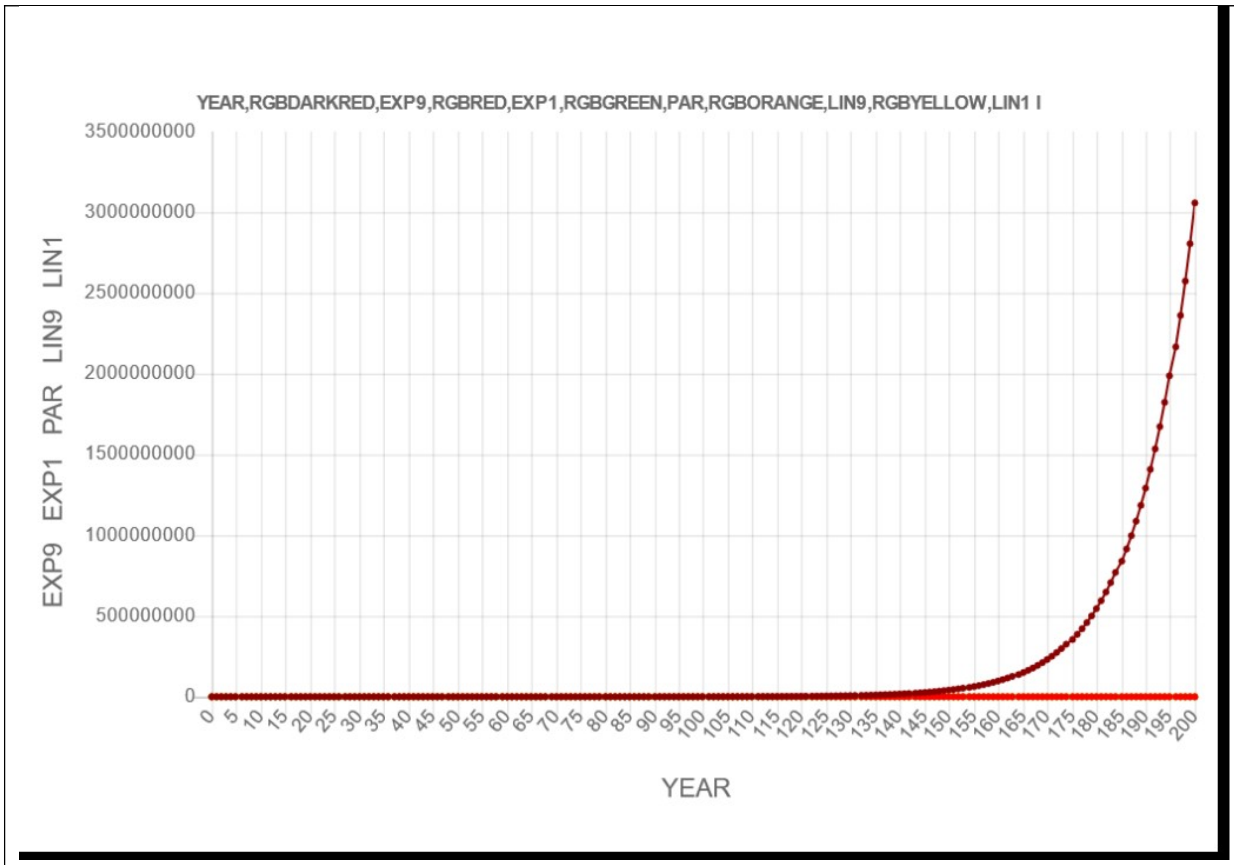
Program 3.3: How does an amount of 100 Euro develop with a "simple" and normal interest rate of 1 % and 9 % and with quadratic growth within 200 years.

```
YEARl := 0 .. 200
#EXP9  :=  100. next EXP9 pred +% 9 at YEAR
EXP1  :=  100. next EXP1 pred +% 1 at YEAR #EXP9
PAR   := YEAR * YEAR + 100
LIN9  :=  100. next LIN9 pred + 9 at PAR
LIN1  :=  100. next LIN1 pred + 1 at LIN9
rnd 0
#sel  YEAR rest 10 = 0 this condition was applied to reduce the volume of
#tab output.
YEAR::=YEAR text
'3
#RGBDARKRED :=darkred leftat EXP9
RGBRED     :=red    leftat EXP1
RGBGREEN   :=green  leftat PAR
RGBORANGE  :=orange leftat LIN9
RGBYELLOW  :=yellow leftat LIN1
```

Result (line chart without EXP9)



Result (line chart with EXPO9)



Result (tab output):

```
YEAR ,EXP9            ,EXP1 ,PAR    ,LIN9   ,LIN1   l

0                100. 100.     100   100. 100.
10               237. 110.     200   190. 110.
20               560. 122.     500   280. 120.
30             1'327. 135.   1'000   370. 130.
40             3'141. 149.   1'700   460. 140.
50             7'436. 164.   2'600   550. 150.
60            17'603. 182.   3'700   640. 160.
70            41'673. 201.   5'000   730. 170.
80            98'655. 222.   6'500   820. 180.
90           233'553. 245.   8'200   910. 190.
100          552'904. 270.  10'100 1'000. 200.
110        1'308'925. 299.  12'200 1'090. 210.
120        3'098'702. 330.  14'500 1'180. 220.
130        7'335'754. 365.  17'000 1'270. 230.
140       17'366'396. 403.  19'700 1'360. 240.
150       41'112'576. 445.  22'600 1'450. 250.
160       97'328'419. 491.  25'700 1'540. 260.
170      230'411'765. 543.  29'000 1'630. 270.
180      545'468'442. 600.  32'500 1'720. 280.
190    1'291'322'174. 662.  36'200 1'810. 290.
200    3'057'029'208. 732.  40'100 1'900. 300.
```

The green parabola is not visible in the second image. The corresponding points are behind the other non-dark red points. Therefore, the parabola looks like a straight line here. The straight line turns into a fast-growing curve when the even faster growing dark red exponential curve is taken out of the picture. This can only be understood by comparing the scapings of the ordinates (Y-axes).

Program 3.4: The chess board problem: Place a grain of wheat on the first square, two on the second, 4 on the third, then eight, and so on. This exponential growth is compared with the polynomial X^8 .

```
Xl    := 1 .. 64
FIELD := 1  next  FIELD pred *2 at X
HIGH8 := X ^ 8
FIELD::= FIELD if X<30 ! (FIELD div 1'000'000)
HIGH8::= HIGH8 if X<30 ! (HIGH8 div 1'000'000)
'3
```

Result (tab):

```
X ,FIELD            ,HIGH8         l
 1                 1               1
 2                 2             256
 3                 4           6'561
 4                 8          65'536
 5                16         390'625
 6                32       1'679'616
 7                64       5'764'801
 8               128      16'777'216
 9               256      43'046'721
10               512     100'000'000
11             1'024     214'358'881
12             2'048     429'981'696
13             4'096     815'730'721
14             8'192   1'475'789'056
15            16'384   2'562'890'625
16            32'768   4'294'967'296
17            65'536   6'975'757'441
18           131'072  11'019'960'576
19           262'144  16'983'563'041
20           524'288  25'600'000'000
21         1'048'576  37'822'859'361
22         2'097'152  54'875'873'536
23         4'194'304  78'310'985'281
24         8'388'608 110'075'314'176
25        16'777'216 152'587'890'625
26        33'554'432 208'827'064'576
27        67'108'864 282'429'536'481
28       134'217'728 377'801'998'336
29       268'435'456 500'246'412'961
30               536         656'100
31             1'073         852'891
32             2'147       1'099'511
33             4'294       1'406'408
34             8'589       1'785'793
35            17'179       2'251'875
36            34'359       2'821'109
```

```
37            68'719       3'512'479
38           137'438       4'347'792
39           274'877       5'352'009
40           549'755       6'553'600
41         1'099'511       7'984'925
42         2'199'023       9'682'651
43         4'398'046      11'688'200
44         8'796'093      14'048'223
45        17'592'186      16'815'125
46        35'184'372      20'047'612
47        70'368'744      23'811'286
48       140'737'488      28'179'280
49       281'474'976      33'232'930
50       562'949'953      39'062'500
51     1'125'899'906      45'767'944
52     2'251'799'813      53'459'728
53     4'503'599'627      62'259'690
54     9'007'199'254      72'301'961
55    18'014'398'509      83'733'937
56    36'028'797'018      96'717'311
57    72'057'594'037     111'429'157
58   144'115'188'075     128'063'081
59   288'230'376'151     146'830'437
60   576'460'752'303     167'961'600
61 1'152'921'504'606     191'707'312
62 2'305'843'009'213     218'340'105
63 4'611'686'018'427     248'155'780
64 9'223'372'036'854     281'474'976
```

You can see that the polynomial on the sixth field has already exceeded the million, but the exponential function is only at 32. In the last line, on the other hand, it becomes clear that the exponential function is larger than the polynomial value by a factor of about 10'000. From position 30 we omitted the last 6 digits to improve the comparability of such large numbers.

Program 3.5: Calculate the total growth of the gross domestic product in West Germany, East Germany, and China in the years from 1992 to 2014 using the growth data given.

```
<TAB!
YEAR, BRDWA, DDRWA, CHINAWA l
1988   0.       0.     0.
1989   3.9      1.85   4.2
1991  11.09   -47.8   13.56
1992   1.7      6.2   14.3
1993  -2.6      8.7   13.9
1994   1.4      8.1   13.1
1995   1.4      3.5   11.
1996   0.6      1.6    9.9
1997   1.5      0.5    9.2
1998   2.3      0.2    7.8
1999   2.1      1.8    7.6
2000   3.1      1.2    8.4
2001   1.1     -0.6    8.3
2002   0.1      0.2    9.1
2003  -0.1     -0.3   10.
```

```
2004   1.6     1.3  10.1
2005   0.8    -0.2  11.3
2006   3.8     3.4  12.7
2007   3.3     2.9  14.2
2008   1.      0.6   9.6
2009  -6.1    -3.9   9.2
2010   4.3     3.2  10.6
2011   3.8     1.9   9.5
2012   0.4     0.6   7.7
2013   0.1    -0.1   7.7
2014   1.6     1.4   7.4
!TAB>
sel  YEAR>1991
DDR,BRD,CHINA:=100.,100.,100. next preds +%(DDRWA,BRDWA,CHINAWA)
               at CHINAWA
rnd 1
gib YEAR,DDR,BRD,CHINA l
```

Result (tab output):

```
YEAR ,DDR   , BRD   , CHINA  l
1992  100.0  100.0  100.0
1993  108.7   97.4  113.9
1994  117.5   98.8  128.8
1995  121.6  100.1  143.0
1996  123.6  100.7  157.1
1997  124.2  102.3  171.6
1998  124.4  104.6  185.0
1999  126.7  106.8  199.0
2000  128.2  110.1  215.8
2001  127.4  111.3  233.7
2002  127.7  111.4  254.9
2003  127.3  111.3  280.4
2004  128.9  113.1  308.8
2005  128.7  114.0  343.7
2006  133.1  118.3  387.3
2007  136.9  122.3  442.3
2008  137.7  123.5  484.8
2009  132.4  115.9  529.3
2010  136.6  120.9  585.5
2011  139.2  125.5  641.1
2012  140.0  126.0  690.4
2013  139.9  126.2  743.6
2014  141.9  128.2  798.6
```

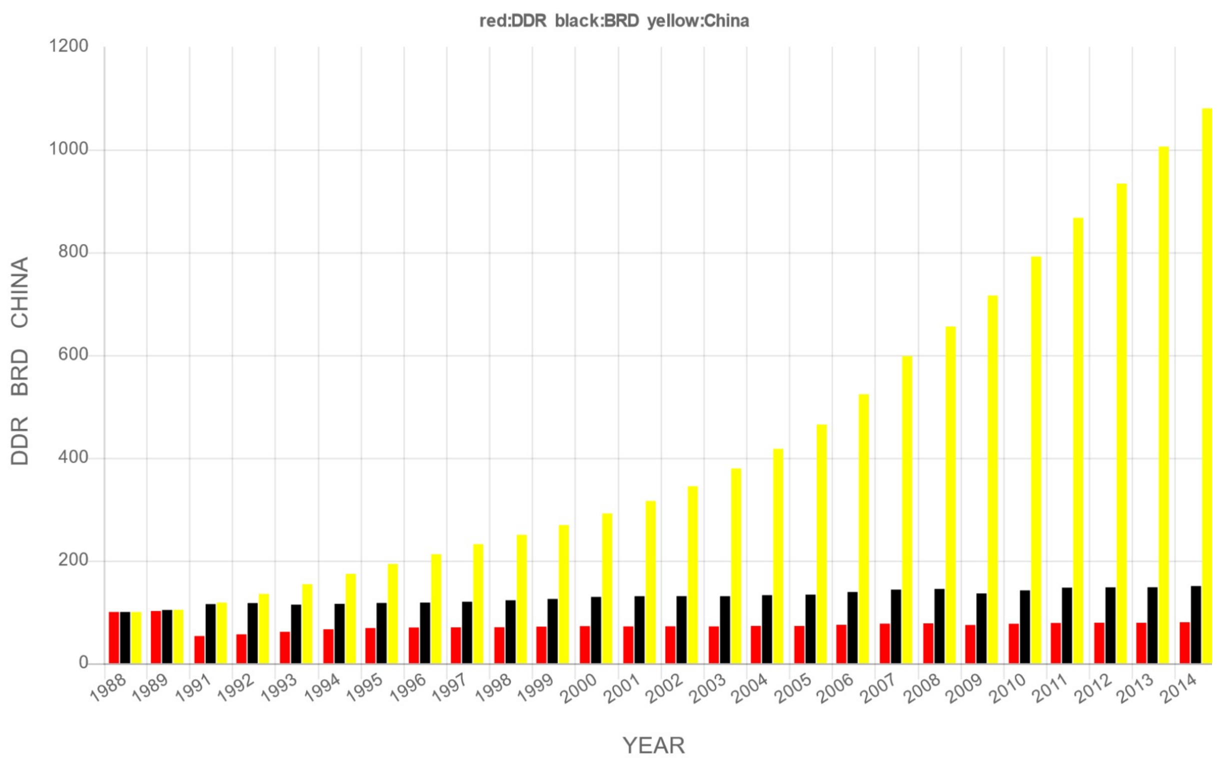
With a total growth of 100 to 142, East Germany is clearly better in this time interval than West Germany with a growth of 100 to 128. Now, we drop the condition YEAR>1991. Furthermore, we assume that the above data enclosed in TAB brackets are in the file growth.tab.

Program 3.6: Calculate the growth of the gross domestic product in East Germany, West Germany and China in the years 1988 to 2014 with the indicated growth.

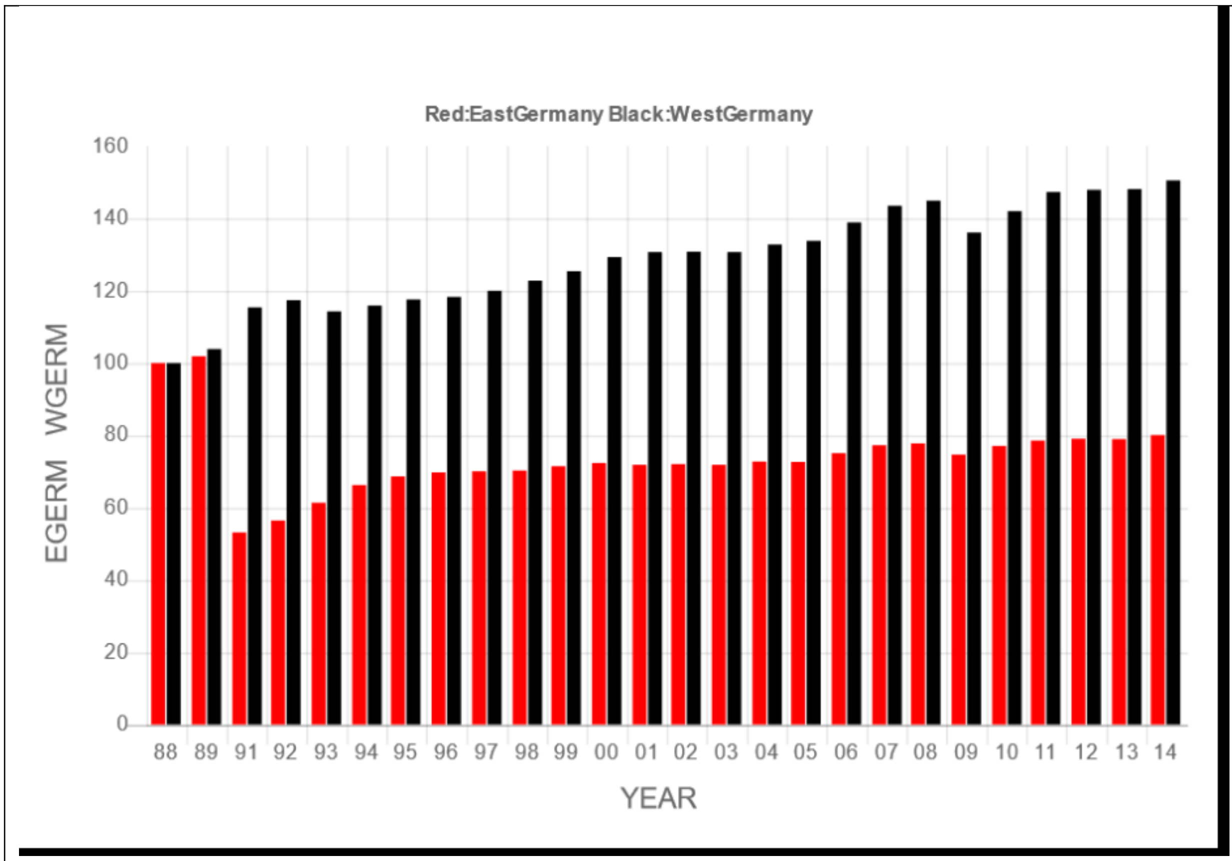
```
growth.tab
DDR,BRD,CHINA:=100.,100.,100.  next  preds +% (DDRWA,BRDWA,CHINAWA)
               at CHINAWA
rnd 1
YEAR::= YEAR text #subtext 3!2
```

```
TITEL:="red:DDR  black:BRD  yellow:China"
gib TITEL,(YEAR,DDR,BRD,CHINA l)
RGB:=red    leftat DDR
RGB:=black  leftat BRD
RGB:=yellow leftat CHINA
```

Result (bar chart):



Result excluding China (bar chart):



Result (tab output):

```
TITEL      ,(YEAR ,RGB     ,DDR ,  RGB     ,BRD ,  RGB     ,CHINA  1)
Red:EastG... 88    1.,0.,0. 100.0  0.,0.,0. 100.0  1.,1.,0.  100.0
             89    1.,0.,0. 101.9  0.,0.,0. 103.9  1.,1.,0.  104.2
             91    1.,0.,0.  53.2  0.,0.,0. 115.4  1.,1.,0.  118.3
             92    1.,0.,0.  56.5  0.,0.,0. 117.4  1.,1.,0.  135.3
             93    1.,0.,0.  61.4  0.,0.,0. 114.3  1.,1.,0.  154.1
             94    1.,0.,0.  66.3  0.,0.,0. 115.9  1.,1.,0.  174.2
             95    1.,0.,0.  68.7  0.,0.,0. 117.6  1.,1.,0.  193.4
             96    1.,0.,0.  69.8  0.,0.,0. 118.3  1.,1.,0.  212.5
             97    1.,0.,0.  70.1  0.,0.,0. 120.0  1.,1.,0.  232.1
             98    1.,0.,0.  70.3  0.,0.,0. 122.8  1.,1.,0.  250.2
             99    1.,0.,0.  71.5  0.,0.,0. 125.4  1.,1.,0.  269.2
             00    1.,0.,0.  72.4  0.,0.,0. 129.3  1.,1.,0.  291.8
             01    1.,0.,0.  71.9  0.,0.,0. 130.7  1.,1.,0.  316.1
             02    1.,0.,0.  72.1  0.,0.,0. 130.8  1.,1.,0.  344.8
             03    1.,0.,0.  71.9  0.,0.,0. 130.7  1.,1.,0.  379.3
             04    1.,0.,0.  72.8  0.,0.,0. 132.8  1.,1.,0.  417.6
             05    1.,0.,0.  72.7  0.,0.,0. 133.8  1.,1.,0.  464.8
             06    1.,0.,0.  75.1  0.,0.,0. 138.9  1.,1.,0.  523.8
             07    1.,0.,0.  77.3  0.,0.,0. 143.5  1.,1.,0.  598.2
             08    1.,0.,0.  77.8  0.,0.,0. 144.9  1.,1.,0.  655.6
             09    1.,0.,0.  74.7  0.,0.,0. 136.1  1.,1.,0.  715.9
             10    1.,0.,0.  77.1  0.,0.,0. 142.0  1.,1.,0.  791.8
             11    1.,0.,0.  78.6  0.,0.,0. 147.3  1.,1.,0.  867.1
             12    1.,0.,0.  79.1  0.,0.,0. 147.9  1.,1.,0.  933.8
             13    1.,0.,0.  79.0  0.,0.,0. 148.1  1.,1.,0. 1005.7
             14    1.,0.,0.  80.1  0.,0.,0. 150.5  1.,1.,0. 1080.2
```

We have hidden China in the second chart so that it is easier to see Germany's two growth data. For example, East Germany produces less than in GDR times. The banking crisis had a major negative impact on the East German economy, even though East Germany does not have a bank, ... Too much information can obscure what seems to be essential.

4 Hello otto - gimmick

Program 4.1: Output two words.

```
Hello otto
```

Result (tabh)

```
WORT1
Hello otto
```

Program 4.2: Output a pair of two words.

```
Hello, otto
```

Result (tab)

```
WORT, WORT
Hello otto
```

Program 4.3: Output a text with spaces.

```
"Hello Otto"
```

Result (tab)

```
TEXT
Hello otto
```

Program 4.4: Concatenate two words with spaces.

```
Hello + " " + otto
```

Result (tab)

```
TEXT
Hello otto
```

Program 4.5: Give a greeting with a list of two words.

```
GREETING := Hello otto
```

Result (ment)

```
TABMENT! GREETING
GREETING! WORT1
```

```
<GREETING>
Hello
otto
</GREETING>
```

Program 4.6: Output two words each with its own column name.

```
DEAR:=Hello
GREETING:=otto
```

Result (tab)

```
DEAR, GREETING
Hello otto
```

Program 4.7: Output two words with one column name.

```
GREETING:= "Hello otto"
```

Result (tabh)

```
GREETING
Hello otto
```

Program 4.8: Sort a set of words.

```
GREETING:= {otto Hello}
```

Result (tabh)

```
TABMENT! GREETING
GREETING! WORTm
GREETING
Hello otto
```

Program 4.9: Represent one word by metadata and the other by primary data.

```
HELLO := otto
```

Result (tab)

```
HELLO
otto
```

5 o++o for kindergarten?

The stroke list is historically the first representation of a number. It could already be a million years old. Notched wood has been shown to be 150 thousand years old. Concepts first developed in history are usually simpler than later concepts. That's why tally charts should have a broader scope even in kindergarten.
The following goals could be pursued with the use of o++o in kindergarten:

1. By presenting decimal numbers and stroke lists at the same time, a child can better appreciate the magnitude of numbers. For example, the number one hundred differs from the number ten only by one digit zero. The corresponding stroke lists, however, differ considerably.
2. The operation symbols + * - : could be taught. They are probably easier to explain on stroke lists. The stroke lists could be converted to decimals and vice versa.
3. The algorithm behind the stroke list operation (gib statement) could be taught using appropriate examples.

5.1 Stroke Lists

Counting animals of different species could give the following small intermediate table:

```
Elephant | | | |
Deer     | |
Pig      | | |
```

If another deer comes, a stroke is added to the second line. If, on the other hand, a turkey comes, a new line must be added with the name turkey and a stroke at the end.

This already poses many problems, although preschoolers can already create such a table if the words have been replaced by pictures or single letters. It is not clear how many columns this table has if only "normal" tables are considered. If we allow structured tables, we can say that this table contains a column ANIMAL and a column STROKE, but the values of the column STROKE can be "repeated" for each animal. An associated schema `ANIMAL,STROKEl m` or

`ANIMAL,STROKEl l` would express this. Where l is an abbreviation for list and m stands for set. These symbols are again used postfix, i.e. they are placed after its arguments. The m is necessary in a gib-part so that each animal appears only once in the target table.

Since many children are interested in cars, one could count cars analogously to counting animals. This could result in the following table:

```
Golf     | | | |
A6       | | |
Polo     | | | | |
Wartburg | | |
A8       | |
```

Is it possible in kindergarten to increase the structural depth of the table when counting? Then the following (hsqh-) table could have been created:

```
VW | | | | | | | | |
 Golf | | | |
 Polo | | | | |
Audi | | |
 A6 | | |
 A8 | |
IFA | | |
 Wartburg | | |
```

5.2 The conversion operations stroke list to zahl and vice versa

As a dash (stroke) o++o uses the "|" character. We will illustrate the operations in the following text with self-explanatory examples.

Program 5.2.1: Stroke list to number	Result
`			zahl`	3

Convert a number into a list of strokes:

Program 5.2.2: Number to tally list	Result
`| *l 4`	`|		|`

5.3 The operations + *

Different representations of an addition task. The first input type again determines the output type.

Programs 5.3.1: Four plus four	Results
`4 + 4`	8
`4 +				`	8
`|			+			|`	`|						|`
`				+ 4`	`								`
`	*l (4 + 4)`	`								`

Different representations of a multiplication task:

Programs 5.3.2: Ten times ten	Results
`10 * 10`	100
`Xl:= 1 ..10` `Yl:=	*l 10 at X`	` 1									|` ` 2									|` ` 3									|` ` 4									|` ` 5									|` ` 6									|` ` 7									|` ` 8									|` ` 9									|` `10									|`

Programs 5.3.3: Two representations of a subtraction task	Result
`10 - 5`	5
`|									- 5`	`|			|`

Program 5.3.4: Place next to each number smaller	Result

than 16 the corresponding stroke list	

Xl:=0 ..15
Y := | *l X
	0
	1 |
	2 | |
	3 | | |
	4 | | | |
	5 | | | | |
	6 | | | | | |
	7 | | | | | | |
	8 | | | | | | | |
	9 | | | | | | | | |
	10 | | | | | | | | | |
	11 | | | | | | | | | | |
	12 | | | | | | | | | | | |
	13 | | | | | | | | | | | | |
	14 | | | | | | | | | | | | | |
	15 | | | | | | | | | | | | | | |

5.4 o++o programs to kindergarten?

Which of the following programs are useful for understanding and which are teachable? When is a syntax too incomprehensible? These questions are outlined below.

Multiplication is counting the number of strokes in a rectangle?

Program 5.4.1: Each of four children gets 3 apples. How many apples are there in total ?	Result (tabh)

NAMEl := Ernst Clara Sophia Claudia
APPLEl:= | | | at NAME
++
	Intermediate result after the first 2 lines

	NAME, APPLE l Ernst | | | Clara | | | Sophia | | | Claudia | | |
	Final result (++ stands for many additions)
	12

Program 5.4.2: Each of four children gets 3 apples. How many apples are there in total?	Result (tabh)

NAMEl := Ernst Clara Sophia Claudia
APPLEl:= | | | at NAME
gib APPLEl
++1
	Intermediate result after the first 2 lines

	NAME, APPLE l Ernst | | | Clara | | | Sophia | | | Claudia | | |
	Final result (++1 counts)
	12

Program 5.4.3: Counting different kinds of animals
ANIMALl:=elephant deer elephant pig elephant deer pig elephant STROKE:= | gib ANIMAL,STROKEl m

Result (tabh)

```
ANIMAL   ,STROKEl  m
deer       | |
elephant | | | |
pig        | | |
```

Program 5.4.4: Counting cars

```
<TAB!
BRAND,COLOR,  TYPE, WEIGHT l
VW    Blue    Polo   1250
IFA   Papyrus 500     580
VW    Blue    Golf   1450
Audi  Yellow  Quatro 2070
VW    Blue    Polo   1380
IFA   Beige   601     620
VW    Red     Golf   1400
Audi  Red     Quatro 2100
IFA   Beige   601     620
VW    Beige   Polo   1300
!TAB>
gib BRAND,CNT,(COLOR,CNT m) m
    CNT:=TYPE ! ++|
```

Result (tabh)

```
BRAND,CNT,     (COLOR,  CNT l) l
Audi  | |        Yellow  |
                 Red     |
IFA   | | |      Beige   | |
                 Papyrus |
VW    | | | | |  Beige   |
                 Blue    | | |
                 Red     |
```

Programs 5.4.5: 3 Division Operations	Results
`13 div 4`	`3`
`13 : 4`	`3.25`
`13 divrest 4`	`3,1`

All these operations seem too complicated for kindergarten.
If one calculates not only with numbers but also with tables, one could introduce new division operations. However, this cannot be discussed to the end at this point.

Program 5.4.6: Problem: Distribute 15 apples among 4 children. Who designs the o++o program?	Result (tabh)

	`Ernst   | | | |`
	`Clara   | | | |`
	`Sophia  | | | |`
	`Claudia | | |`

Another very important operation of digitization is selection. Would a database operation like selection be teachable to some degree?

given:

```
NAME,    AGE l
Ernst    8
Clara    6
Sophia   6
Claudia  4
Ulrike   5
Käthe    4
```

myfamily.tab

Program 5.4.7: How old is Claudia?	Result

`aus myfamily.tab`
`sel Claudia`
	`NAME,    AGE l`
	`Claudia 4`

`sel abbreviates selection.`

Program 5.4.8: All six years old children are wanted	Result

`aus myfamily.tab`
`sel AGE = 6`
	`NAME,   AGE l`
	`Clara   6`
	`Sophia  6`

Program 5.4.9: All children younger than 6 are wanted	Result

`myfamily.tab`
`sel AGE < 6`
	`NAME,    AGE l`
	`Claudia 4`
	`Ulrike  5`
	`Käthe   4`

6 o++o in School Lessons

There are many possible applications for o++o in school. Especially in the subjects, mathematics and computer science. But, also in all other subjects o++o can be used to extract data from given tables, documents. We do not want to present all possible typical query examples here. We want to limit ourselves to the so-called "brute force algorithms" for mathematics. These are the simplest and often the methodologically best algorithms. Since all these algorithms are implemented in main memory, we need not worry about efficiency. Now we start with a simple algorithm. We hope that it is the simplest program for a zero. The section ends with programs for grading students and considerations that may be important for kindergarten, too.

Program 6.1: Calculate in a simple way the zero of the sine function in the interval [3, 4].	Result
`Xl:= 3 ... 4! 0.000’01`	`Xl`
`sel  X sin <0`	`3.1416`
`sel  X pos =1`	

Program 6.2: Calculate in a simple way the zero of the sine function in the interval [3, 4].	Result
`Xl:= 3 ... 4! 0.000’01`	`Xl`
`sel  X sin * (X +0.000’01 sin) <=0`	`3.14159`

Program 6.3: Calculate the integer zeros of the polynomial "X^2 -15X+56".	Result (tabh)
`Xl:= -100 .. 100`	`Xl`
`sel  X poly [1 -15 56]=0`	`7 8`
`# or sel  X- 15 *X + 56=0`	

With the following programs, it is shown that students who have not learned integral and differential calculus are nevertheless able to understand and use their school applications, which are essentially:

1. How large are areas under curves?
2. What are local extrema of functions?

Program 6.4: Calculate the area under a circular arc with diameter 4 in the interval [0, 2].	Result
`Xl:= 0 ... 2!0.0001`	`AGG`
`HEIGHT:= X*X - 4 abs sqrt`	`3.14169223791`
`RECTANGLE:=HEIGHT*0.0001`	
`gib AGG   AGG:=RECTANGLE!++`	

Program 6.5: Query 6.4, but shorter and more precise.
`PI:=0 ... 2!0.000'001 poly [-1 0 4] sqrt*0.000'001 ++ '3`
Result
`PI`
`3.141'593'653'28`

Program 6.6: Determine pi by zero determination with interval bisection
`NR,LE,RI l := 3.,4. while NR <39 ! preds ++ :2,RI pred`
`              if preds ++ :2 sin >0 ! LE pred, (preds ++ :2)`

```
last
PI:=pi
'3
```

Intermediate result after first and last program line (tab):

```
NR, LE ,              RI l
 1  3.                4.
 2  3.                3.5
 3  3.                3.25
 4  3.125             3.25
 5  3.125             3.187'5
 6  3.125             3.156'25
 7  3.140'625         3.156'25
 8  3.140'625         3.148'437'5
 9  3.140'625         3.144'531'25
10  3.140'625         3.142'578'125
11  3.140'625         3.141'601'562'5
12  3.141'113'281'25 3.141'601'562'5
13  3.141'357'421'88 3.141'601'562'5
14  3.141'479'492'19 3.141'601'562'5
15  3.141'540'527'34 3.141'601'562'5
16  3.141'571'044'92 3.141'601'562'5
17  3.141'586'303'71 3.141'601'562'5
18  3.141'586'303'71 3.141'593'933'11
19  3.141'590'118'41 3.141'593'933'11
20  3.141'592'025'76 3.141'593'933'11
21  3.141'592'025'76 3.141'592'979'43
22  3.141'592'502'59 3.141'592'979'43
23  3.141'592'502'59 3.141'592'741'01
24  3.141'592'621'8  3.141'592'741'01
25  3.141'592'621'8  3.141'592'681'41
26  3.141'592'651'61 3.141'592'681'41
27  3.141'592'651'61 3.141'592'666'51
28  3.141'592'651'61 3.141'592'659'06
29  3.141'592'651'61 3.141'592'655'33
30  3.141'592'653'47 3.141'592'655'33
31  3.141'592'653'47 3.141'592'654'4
32  3.141'592'653'47 3.141'592'653'93
33  3.141'592'653'47 3.141'592'653'7
34  3.141'592'653'58 3.141'592'653'7
35  3.141'592'653'58 3.141'592'653'64
36  3.141'592'653'58 3.141'592'653'61
37  3.141'592'653'58 3.141'592'653'6
38  3.141'592'653'58 3.141'592'653'59
```

Final result:

```
NR, LE,              RI  l,           PI
38  3.141'592'653'58 3.141'592'653'59 3.141'592'653'59
```

Program 6.7: Calculate the first 7 Fibonacci numbers.

```
Xl:= 1 .. 7
FIB1,FIB2:= 0,1 next FIB2 pred; preds ++ at X
```

Result (tab)

```
X, FIB1, FIB2 l
1  0       1
```

```
2  1      1
3  1      2
4  2      3
5  3      5
6  5      8
7  8     13
```

Program 6.8: Calculate the Pascal triangle up to the exponent 9.

```
Nl:= 0 .. 9
XTUP:= 1 next 0,preds + (preds,0) at N
text
```

Result (tab)

```
N, XTUP l
0  1
1  1,1
2  1,2,1
3  1,3,3,1
4  1,4,6,4,1
5  1,5,10,10,5,1
6  1,6,15,20,15,6,1
7  1,7,21,35,35,21,7,1
8  1,8,28,56,70,56,28,8,1
9  1,9,36,84,126,126,84,36,9,1
```

Now we present a brute force algorithm for a maximum.

Program 6.9: Find in a simple way for the local maximum of the sine function in the interval [1, 3].

```
LOCMAX:=1 ... 3.!0.000’01 sin max '3
```

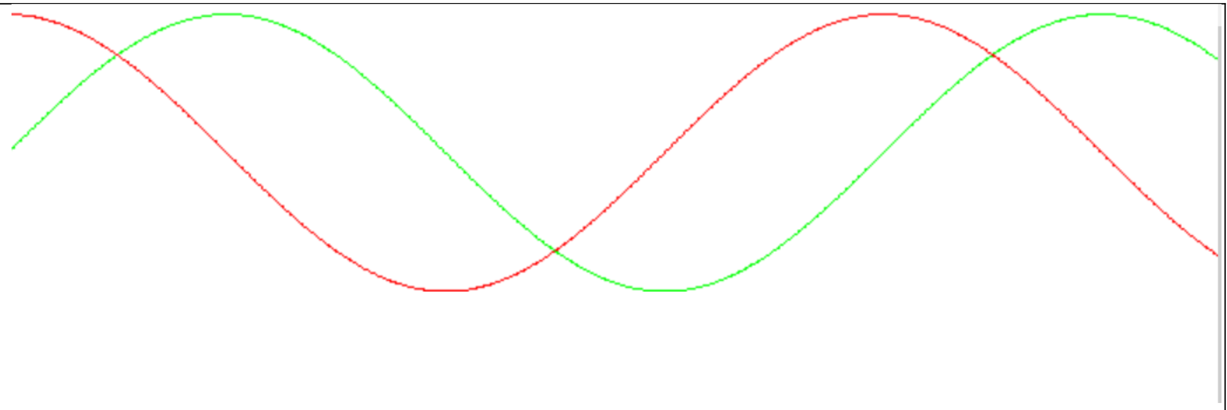
Result

```
LOCMAX
0.999'999'999'993
```

Program 6.10: Calculate the sine function and an approximation of the first derivative in the interval [0,4].

```
Xl:= 0 ... 10!0.01
SINUS := X sin
DERIVATIVE:=X+0.000'1 sin -(X sin):0.000'1
RGBSIN:=green leftat SINUS
RGBDERIVATIVE:=red leftat DERIVATIVE
```

Result (image):



Result (tab): It consists of 1001 lines.

```
X     ,RGBSIN   ,SINUS                ,RGBDERIVATIVE ,DERIVATIVE       l
 0.   0.,1.,0.  0.                    1.,0.,0.        0.999999998333
 0.01 0.,1.,0.  0.00999983333417 1.,0.,0.        0.999949498758
 0.02 0.,1.,0.  0.0199986666933  1.,0.,0.        0.999799005067
 0.03 0.,1.,0.  0.0299955002025  1.,0.,0.        0.999548532308
 0.04 0.,1.,0.  0.0399893341866  1.,0.,0.        0.999198105529
 0.05 0.,1.,0.  0.0499791692707  1.,0.,0.        0.998747759772
 0.06 0.,1.,0.  0.0599640064794  1.,0.,0.        0.998197540071
 0.07 0.,1.,0.  0.0699428473375  1.,0.,0.        0.997547501448
 0.08 0.,1.,0.  0.0799146939692  1.,0.,0.        0.996797708907
 0.09 0.,1.,0.  0.089878549198   1.,0.,0.        0.995948237425
 0.1  0.,1.,0.  0.0998334166468  1.,0.,0.        0.994999171949
...
 9.98 0.,1.,0. -0.527131998452   1.,0.,0.       -0.849757059217
 9.99 0.,1.,0. -0.535603334614   1.,0.,0.       -0.844442914713
10.   0.,1.,0. -0.544021110889   1.,0.,0.       -0.83904432662
```

The following examples are based on a fictitious table of grades with exams:

```
NAME,      (SUBJECT,    EXAl,MARKl l)l
Einstein    German      1 3  1 2 1 3 1
            Physics     1 a  1 2 1 1 1 1
            Algebra     1 2  1 1 2
            Art         3 3  2 1
Gauss       German      2 3  1 2
            Algebra     1 1  1 1 1
Guericke    physics     s 1  1 2 1
            German      2 1  1 2 1 1
            Algebra     1 1  2 1 1 2
Newton      Physics     1 1  2 1 1
Confucius   Philosophy 1 1  1 1 1 1
            Chinese     1 1  2 1 1
Marx        economics   1 1  2 1 2 1
            Philosophy 1 2  1 3 1
Brecht      German      1 1  1 1 1 1 1 1 1 1
            Philosophy 1 2  2 1 1 2
Cantor      set theory 1 1  1 1 1
```

Tabment 6.1: guys.tabh (s: sick, a: absent)

Program 6.11: Calculate the weighted average scores for each person and subject and the total value. Sort the data.

```
aus guys.tabh
gib AVG,(NAME,AVG,(SUBJECT,AVG m)m)
    AVG:=EXAl ++: *0.6 + (MARKl ++: *0.4)! ++:
rnd 1
```

Result (tab):

```
AVG ,(NAME      ,AVG  ,(SUBJECT   ,AVG  m) m)
1.4   Brecht    1.3    German      1.0
                       Philosophy  1.5
      Cantor    1.0    set theory  1.0
      Confucius 1.1    Chinese     1.1
                       Philosophy  1.0
      Einstein  1.7    Algebra     1.4
                       Art         2.4
                       German      1.8
                       Physics     1.1
      Gauss     1.6    Algebra     1.0
                       German      2.1
      Guericke  1.2    Algebra     1.2
                       German      1.4
                       physics     1.1
      Marx      1.4    Philosophy  1.6
                       economics   1.2
      Newton    1.1    Physics     1.1
```

If we want to calculate the average after the completion of the first test, we can use the following formula:

```
AVG:= EXAl nth 1 *0.3 + (MARKl ++: *0.7)! ++:
```

Program 6.12: Determine all subjects and individuals that received a 1 and a subsequent 3.

```
aus guys.tabh
sel NAME SUBJECT! MARK=1 & MARK succ=3      #succ = successor
```

Result:

```
NAME,    (SUBJECT,    EXAl, MARKl l)l
Einstein German      1 3    1 2 1 3 1
Marx     Philosophy  1 2    1 3 1
```

Now we turn to other "simple" problems. It may be that these tasks are important not only for school, but also for kindergarten. Now it is commonly assumed that addition of natural numbers is the easiest and division the most difficult of the four basic arithmetic operations. This could be wrong. We did experiments with a 3-year-old and a 6-year-old kindergartener. The task was to divide 11 apples among four children. The four children were represented by photographs. Neither the 6-year-old child nor the 3-year-old child had a problem. They obtained the same result in the division. It was presented in a table:

```
CHILD,   APPLE
Ernst    | | |
Clara    | | |
Sophia   | | |
```

`Claudia | |`

Tabment 6.1: 11 divided by 4

What can we learn from this experiment?

1. Young children cannot divide an apple. They do not yet have a clear understanding of 1/2 or 2/3, ..., so that "ordinary" division cannot be taught.
2. There is no remainder in the division; there is no reason to waste anything.

Let's consider **addition**, the next simplest operation. The simplest representation of the number three are three strokes. The same is true for any number of other natural numbers. Here we consider only two numbers: 3 and 4.
We have to represent them by lists or bags (multisets) because the set {| |} is the same as {|}. The result of each operation would be one.

`APPLE l`
`| | |`

three.tabh

`APPLE l`
`| | | |`

four.tabh

Program 6.13: three plus four	Result

`aus three.tabh,four.tabh`
`gib APPLEl`
	`APPLEl`
	`| | | | | | |`

Here, too, it becomes clear that such a process could require only a small amount of effort in the classroom. But what is the result of 4 apples and 3 pears? Since each pair of two tabments is again a tabment, the result of this "addition" could be a tabment of the type APPLEl,PEARl.

Multiplication can also be handled in a very simple way. Consider the very simple question of how many apples are needed for 4 children if each child wants 3 apples:

Program 6.14: four times 3	Intermediate result after line 2

`CHILDl:= Ernst Clara Sophia Claudia`
`APPLEl:= | | |  at CHILD`
`gib APPLEl`
	`CHILD,  APPLE l`
	`Ernst   | | |`
	`Clara   | | |`
	`Sophia  | | |`
	`Claudia | | |`

Not only this multiplication algorithm is simpler, it also makes it clear that multiplication is essentially calculating the area of a rectangle.
We also obtain the above intermediate result by the following program:

```
CHILDl:= Ernst Clara Sophia Claudia
APPLES:= | | |  at CHILD
gib CHILD,APPLES l # APPLES is atomic, i.e. each apple list is transferred
                   # as one unit
```

If we want to apply the **subtraction operation** to collections with different elements (sets), the subtraction can be expressed by a selection.

Program 6.15: Subtraction (difference) with sets	Result

`NAMEm := {Ernst Clara Ulrike}`
`sel- NAME in {Ulrike Sophia}`
	`NAMEl`

	`Clara Ernst`

We conclude this section with the following statements:

1. The result of our "arithmetic operations" are not numbers, but tables.
2. Dealing with tables is probably easier than dealing with numbers, because the level of abstraction is lower.

7 Multiplication, School and Digitization

In this chapter, it will be shown that the common multiplication algorithm for decimal numbers could and should be supplemented by simpler ones and that, more generally, deep digitization should be pursued. Deep digitization can probably only be implemented through mathematical understanding. Unlike shallow digitization, where the user is usually presented with a computer result by simply clicking a button and is often unsure if that result is correct, deep digitization should allow the user to understand the result in the same way as calculating 132.66 times 453.2 with a calculator. The big difference between today's use of calculators and today's use of powerful computers is that users have spent years learning the single data operations: `+ - * : sin log` etc. Mass data operations are not yet on the curriculum. Selection - sometimes called a filter operation - and operations to merge table contents for restructuring ... we count among the mass data operations. These are not applied to individual numbers, but to possibly very large structured tables that may contain words and text in addition to numbers. If the user has understood such mass data operations and they have been implemented within the framework of a programming language, he can also interpret these results and, in case of doubt, correct, change or improve them.

7.1 Who can multiply in their head?

Incident 1
In a mathematics exam that a second-year pharmacy student from Bologna had to take, the student had to calculate 7 times 8, among other things. The pharmacy student: 59
The algebra professor: But 59 is not an even number. The pharmacy student: 64

Incident 2
Wallerie - an Erfurt kindergarten girl in the large group - is already a student today.
I gave her a task: How many effervescent bottles does a crate with 4 rows contain if there are 5 bottles in each row?
Wallerie thought for a while: Nineteen
Her father - a young engineer: You don't calculate, you guess.

Incident 3
I ask Isabella, a second-grade pupil from Gerwisch: How much is 3 times 4? After a while: Twelve
The father: That took a long time.

From the second incident, I conclude that preschoolers have already understood the essence of multiplication. Of course, it is possible that some preschoolers.... cannot calculate 4 times 5 exactly in their head. But 3 times 4 I would trust any child to do. There is no question that they will never be able to calculate 12 times 13 in this way. In fact, I don't think any human being is capable of calculating 7 times 8 in their head. Older adults have had to calculate(?) the multiplication tables so many times in school that they can only do it by heart and don't remember how they multiplied as a child. It used to be very important to know the multiplication table by heart because it was a prerequisite for written decimal multiplication.
My opinion: In the many school years that the multiplication tables were taught, the original neuron connections or brain cells were "overwritten" and are practically no longer present.

Therefore, the vast majority of adults are not able to perform the original multiplication in their heads. They can only do multiplication tables by heart and cannot do written decimal multiplication in their heads. Even when multiplying smaller numbers such as 29 times 63, they will work with easier-to-use arithmetic laws and not use the algorithm in which their teachers, parents and grandparents invested a lot of time and effort. Based on incidents 1, 2, and 3, one can even surmise that many adults don't even know that children have to do math to get the results. When you're that young, you can't memorize it yet without doing the math. An almost correct answer indicates that arithmetic has been done.

7.2 Who can multiply in writing?

Calculating `7 times 8` with a pencil should be mastered by every child in the second grade. The prerequisite is that you can imagine the numbers up to one hundred. You can do that if you can count to a hundred. If you illustrate the task, many children should be able to solve it even faster:
Each of the seven children wants eight candies. How many candies do you need to buy?

1. Write the names of seven children one below the other.
2. Put eight strokes legibly after each name.
3. Count all the strokes.
4. Convert the result into a decimal number.

If someone wants to calculate `100 times 100 in` this way, the probability of getting a correct result is very low. Moreover, it would take a very, very long time. In the age of powerful computers, however, these arguments should be insignificant. What matters is to have a clear understanding of an algorithm. The question remains:
Is stroke list multiplication the simplest multiplication algorithm?

7.3 Who can program the multiplication?

Program 7.3.1: Stroke list multiplication in o++o (7*8)

```
NAMEl:=Tina Ernst Clara Sophia Ulrike Claudia Kathe
STROKEl:= | | | | | | | |  at NAME
gib STROKEl
++1
```

Intermediate result after 2 lines in tabh format

```
NAME,   STROKEl  l
Tina    | | | | | | | |
Ernst   | | | | | | | |
Clara   | | | | | | | |
Sophia  | | | | | | | |
Ulrike  | | | | | | | |
Claudia | | | | | | | |
Kathe   | | | | | | | |
```

Final result

```
56
```

If you are only interested in the end result, you can also replace all names with one. You could also write a gib statement for the last two lines that counts through regardless of the given structure:

Program 7.3.2: Shortened stroke list multiplication in o++o

```
NAMEl   := otto *l 7
STROKEl:= | *l 8 at NAME
gib SEVEN_TIME_EIGHT
    SEVEN_TIMES_EIGHT:= STROKE! ++1
```

Result

```
SEVEN_TIMES_EIGHT
56
```

This program can only be formulated because o++o works with structured tables. Everybody should judge for himself which multiplication is more child-like and therefore easier to understand. At this point it should be mentioned that o++o multiplication is much more general than in other programming languages. However, this does not mean the program above, but the operation hidden behind the symbol *. For example, you can multiply a whole tabment by a number:

Program 7.3.3: Convert several German net prices with o++o into gross prices.

```
GROSSl:=66.1 675.8 77 *1.19
```

Result (tabh)

```
GROSSl
78.659 804.202 91.63
```

The section concludes with a multiplication algorithm that is reminiscent of pivot tables, but very close to the decimal multiplication algorithm taught in schools today.
The pivot element can be determined with 2 o++o lines.

Program 7.3.4: Multiplication of a list of length 2 by a triple using matrix multiplication (13*124)

```
[10 3] *mat (100,20,4)
cross ++
```

Intermediate result after the first line in tab format

```
ZAHL ,ZAHL ,ZAHL  l
1000  200   40
 300   60   12
```

Final result in (.tab)

```
ZAHL ,ZAHL ,ZAHL ,SUM?  l
1000  200   40    1240
 300   60   12     372
1300  260   52    1612
```

The final result of multiplying `13 times 124` is `1612.`
Math teachers must find out whether this multiplication is easier to teach than today's multiplication with decimals by having entire classes multiply using both methods.

Program 7.3.5: A complete o++o-program for multiplication, based on matrix-multiplication

```
X:=4321
Y:=678
XX:=X cut 1 zahl
YY:=Y cut 1 zahl
XL:=XX ++1
YL:=YY ++1
BXl:= 10 ^ (XL- 1 ... 0! -1) * XX
BYl:= 10 ^ (YL- 1 ... 0! -1) * YY
MATRIX:= BXl *mat (BYl transpose)
gib MATRIX
cross ++
last
gib SUM
'3
```

Result (tab)

```
SUM
2'929'638
```

cut and other operations could also be used advantageously for teaching German or English. Each letter is converted into a digit by the conversion function `zahl`. Teaching cross-multiplication in one form or another would also prepare students for the use of pivot tables, which play a large role in today's practice.

The last program in this section is intended to implement pivot multiplication with the help of user defined o++o-operations.

Program 7.3.6: More detailed matrix multiplication of factors 7'653 and 4'322 in o++o

```
defop $X myop.powerlist = begin
X1l:=$X cut 1 zahl
gib X1 l-
XPOT:= (10 ^ (X1 pos - 1))
X2:= XPOT * X1
gib X2l-
end
aus 7'653 myop.powerlist
*mat 4'322 myop.powerlist transpose
cross ++
last
gib SUM
'3
```

Final result (tab)

```
33'076'266
```

7.4 Stroke list multiplication versus decimal multiplication

The written decimal multiplication had a great importance because many people could calculate with it even two 8-digit numbers correctly with high probability. The correctness could be improved even more by the sample of nine. In addition, it was by far the fastest algorithm in earlier practice.
The later slide rule was faster, but not as accurate. The number table with logarithms was too demanding for some. In the age of computers, both techniques have already been mothballed.
The written decimal algorithm is still used by many people ..., and everyone who masters it is proud of his skills. The question is: Can we replace the decimal multiplication algorithm in school with the stroke list multiplication or/and complement it with other multiplication algorithms?
The dash multiplication has not only the advantage that it can be taught already in an earlier class. If this algorithm is repeated accordingly, everybody notices that just this multiplication realizes the standard application - a rectangular area calculation. To derive this application from decimal multiplication seems too difficult. Given the programmability of both algorithms, it should quickly become clear that dash multiplication is far superior to decimal multiplication. The dash list multiplication above requires 4 steps to be processed in sequence. In particular, there is no loop and no recursion. On the other hand, the above algorithm shows that a programming language must be able to work with structured tables in order to provide user-friendly multiplication programs. Since this stroke multiplication processes mass data in a sense, it prepares better for the digitization of society than the algorithm taught in schools today. If it is desired that everyone should be able to program multiplication, simpler multiplication algorithms must be taught.

7.5 How o++o could enrich the school curriculum?

We have developed a data model with an associated programming language o++o, which should not only be the basis for information systems for business, but also offer many advantages for school teaching. Since the language o++o is based on mathematical concepts, it should be integrated into mathematics classes. But, also in the other subjects o++o can be used usefully, because the extraction and visualization of information from the German Wikipedia seems to be important for every school subject. A programming language should enable students to better solve the many tasks they will face in their future. This is especially true for the digitalization ahead of us. Digital actually means that everything comes down to two things - zero and one. I can't imagine anyone tracing the powerful stroke list operation behind the gib statement, for example, to such thinking. Our axioms of the stroke list operation were formulated at a high abstract algebraic level, where thinking in terms of zeros and ones is only a hindrance. For mathematics, abstraction is more important than digitization in the true sense of the word. Although o++o has so far only dealt with questions of content, CSS in o++o can also be used to realize many questions of format. The following example shows that it also makes sense to capture form questions directly in o++o:

Program 7.5.1: The product of 5 numbers in thousands format in o++o

```
28'911 5'233 199 6'311 6'781 ** '3
```

Result

```
1'288'424'128'758'129'267
```

o++o could perhaps be taught in the lower grades:

Program 7.5.2: The sum of the first hundred numbers (Gauss problem)

```
1 .. 100 ++
```

Result

```
5050
```

I think that these and many other difficult problems could be taught in the lower grades as well. So as not to be misunderstood at this point. We should already be drawing on experience gained decades ago with calculators when introducing digitization in schools. Also, the too early and too wide use of calculators has probably led to many students having a poorer command of basic arithmetic..., worse at calculating in their heads than in earlier grades. With incorrect inputs to the calculator, many also seem unable to estimate the expected magnitudes of the results. For this reason, the calculator is not allowed here until seventh grade.
For example, I even think that spelling programs like WORD should not be taught in school until about the seventh grade, either. If a student has experienced firsthand that WORD corrects almost all of his spelling mistakes, it is very difficult to make him understand ... that his own spelling skills are important for his future. Similarly, I would have the introduction of digital whiteboards critically examined.
Last December, at the University of Halle, I noticed that mathematics professors were still working with blackboards and ordinary chalk.

Program 7.5.3: Execute an o++o program in the second class on the blackboard.

```
4 3 1 12 ++
```

This calculation on the blackboard with chalk or pencil could also prepare for future digitization. In addition, for motivation reasons, the teacher could already demonstrate to the lower grade students that the symbol `++ can be` used to solve the Gaussian problem or even larger problems. In my opinion, many people do not actively know how to formulate a conditional, although it is not difficult. However, it is not part of the curriculum. The conditions that select all people living in Magdeburg `LOCATION=Magdeburg` or filter out all rivers that are longer than 1000 km `LENGTH>1000` do not look complicated. Many can't do that, because today's search engines don't ask for that or can't handle it. But if I need to spontaneously extract important information from a company database in a future company, I need to know that.
In my opinion, students' problem-solving skills can be improved in many ways. Even the applications of differential and integral calculus could be taught in secondary schools without having to

understand the difficult theories of Leibniz and Newton. With o++o, we can calculate areas under curves in a short line of code without using hard-to-read loops. An approximation of the area under a part of the first sinusoidal arc can be calculated in one line using Archimedes' 2000 year old algorithm:

Program 7.5.4: This o++o program does not require integral calculus!

```
1 ... 2!0.000’1 sin *0.000’1 ++
```

Result

```
0.956536680039
```

In the following, an application of differential calculus is presented, which can be performed without knowledge of differential calculus.

Program 7.5.5: An o++o program to approximate the local minimum of the parabola (a special polynomial) "3 x^2 + 4 x + 6"!

```
-10 ... 10!0.0001 poly 3 4 6 min
```

Result

```
4.66666667
```

I believe that this can be taught already in every 9th or 10th grade, without going into details here. When I talk to students, I sometimes have the impression that computer science classes are more about form issues (HTML, ...) than content. We know that it is very hard, but we should still reach the goal given by Dr. Angela Merkel: **Everyone should learn to read and calculate, but also to program**. If you look at programming languages like C, Java or Python, the goal is not feasible. For that, you need simpler languages that are able to solve end-user problems with short programs. C and Co. had other goals. They should serve to program systems on which hundreds or more people can work, which can contain many millions of lines of code and still work performantly. o++o follows the new paradigm of table-oriented programming and has above all the goal formulated by A. Merkel. If o++o had not put methodical and pragmatic questions in the foreground from the beginning, this goal would not be realizable also with o++o. Mastering operations for mass data seems to be necessary for a long-term digitization strategy.

7.6 Can the stroke list operation be taught as early as third grade?

As already mentioned, the gib statement, which includes the dash list operation, is a powerful tool. It can be used not only to sort normal flat tables, but also any tables. At the same time, you can also use aggregations such as `++` (sum), `++1` (count), etc. If third grade students have difficulty with a formal syntax, it does not necessarily mean that the algorithm behind it cannot be taught. For example, they could count animals. This does not have to be just a number. A table that determines the number for each type of animal would certainly be easy to teach as well:

Program 7.6.1: Counting animal species with strokes

```
ANIMALl:=donkey sow boar donkey boar sow donkey
STROKE:= | at ANIMAL
gib ANIMAL,STROKEl m
```

Result (tabh)

```
ANIMAL, STROKEl m
```

```
Boar     | |
Donkey   | | |
Sow      | |
```

The following example is a bit more demanding, because the results table is structured.

Program 7.6.2: Counting in structured tables

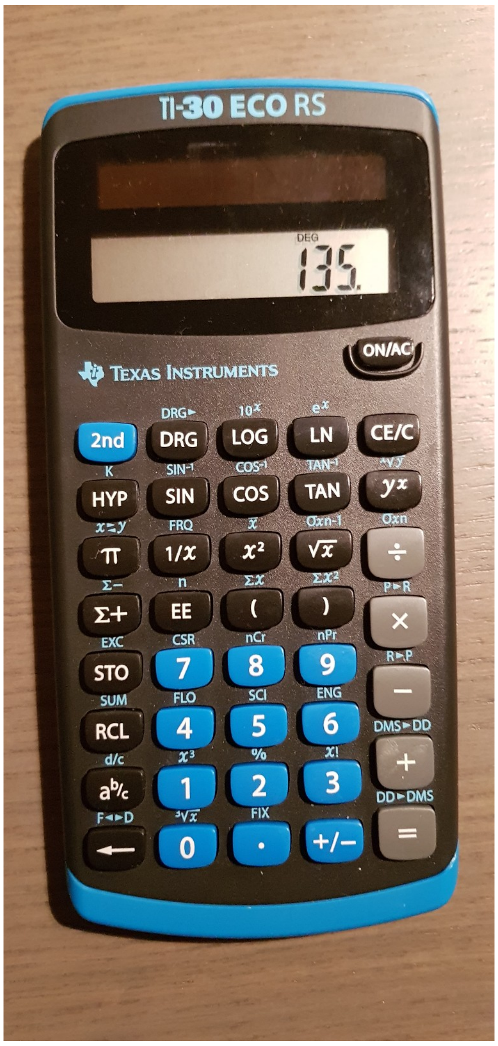
```
<TAB!
BRAND,COLOR,  TYPE,  WEIGHT l
VW    Blue    Polo   1250
IFA   Papyrus 500     580
VW    Blue    Golf   1450
Audi  Yellow  Quatro 2070
VW    Blue    Polo   1380
IFA   Beige   601     620
VW    Red     Golf   1400
Audi  Red     Quatro 2100
IFA   Beige   601     620
VW    Beige   Polo   1300
!TAB>
gib BRAND,CNT,(COLOR,CNT m) m
    CNT:=TYPE! ++|
```

Result:

```
BRAND, CNT,        (COLOR,  CNT m) m
Audi   | |          Yellow  |
                    Red     |
IFA    | | |        Beige   | |
                    Papyrus |
VW     | | | | |    Beige   |
                    Blue    | | |
                    Red     |
```

You can perhaps imagine children counting and sorting at the blackboard using this algorithm. In o++o a set (m) or a multiset (bag) is always sorted by the first column names. In the example above these are `BRAND` and `COLOR`.
That is, children can presumably sort data in structured tables. However, today's computer science students do not learn a sorting algorithm for structured tables. An article of mine in the German Wikipedia, which included especially this sorting, was deleted, because it "does not belong to the basic knowledge of a computer scientist".

7.7 Does the school calculator from Texas-Instruments calculate wrong?



The TI-30 ECO RS calculator shown on the left, which has been approved by German education ministries as a school pocket calculator, gives the following results for the task

2 ^ 2 ^ 3

64. Correct according to the rules of today's mathematical conventions, which can also be read in Wikipedia under operator order (right-associative), would be 256.

For ^, however, you have to type the symbol y^x there.

Now, of course, you can say that every company can calculate as it pleases. They do that, too. With the Windows calculator (mode normal), 1 + 2 × 3 also results in a wrong solution in the sense of school mathematics. Saxony-Anhalt may not have enough money to sue the American tech giant Microsoft. But how can we prevent many students from losing their orientation because of this "diversity"?

As the picture above suggests, the calculator makes a very good impression. However, it behaves differently from what is taught in school in many other aspects and is also difficult to use, making it prone to errors even in simple tasks.

In mathematics, the "sine of 3.14" is usually written as follows:

sin(3,14)

In the mathematics textbook "Schlüssel zur Mathematik" (Sekundarstufe Sachsen-Anhalt Klasse 10 Cornelsen, ISBN 978-3-06-0044558-7) it says more regrettably:

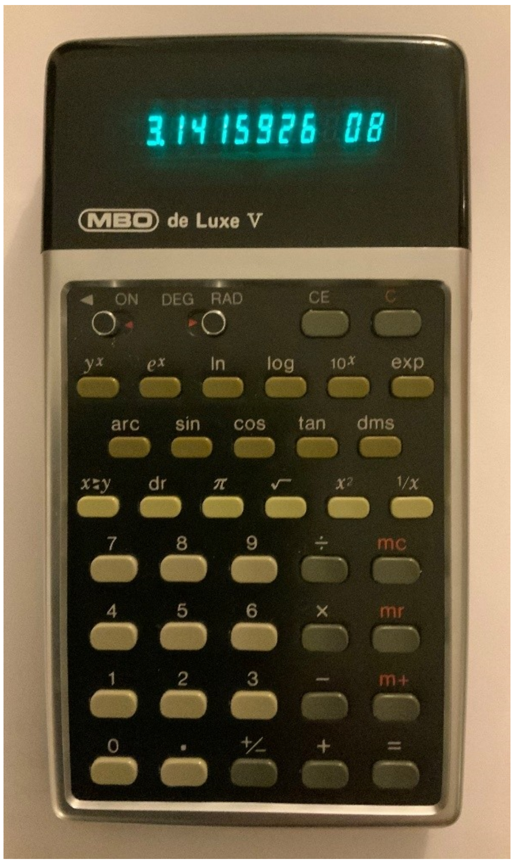
"The function f(x) = sin x is called a sine function."

The Texas Instruments calculator does not accept the comma as a decimal number separator and you must first press 3.14 and then the sin key. At least Texas Instruments is consistent in typing at this point. The square root of 4 is also found by first typing the 4 and then the square root sign. As everyone expects, the result is 2. But 2 without the decimal point would also be conceivable? With 2+2, Texas Instruments also determines 4. and not 4, although everyone knows that the result of this addition is an integer. To prevent misunderstandings at this point: We do not criticize that this Texas Instruments calculator chooses the more user-friendly typing variant for unary operations, but that curriculum and school practice differ substantially here.

Designations on the keyboard are also surprising.

Σ+ (EE, RCL, STO, ...).

Many people are already familiar with the M+ symbol - add to memory - due to predecessor computers. Is innovation to be feigned here?



In this context, it is also interesting to note that pocket calculators already existed in the 1970s whose range of functions was perfectly adequate for use in mathematics lessons and for a large number of applications, especially in the scientific and technical fields.

These calculators were characterized by a clear keyboard layout that did without multiple key assignments. The calculator architecture consistently implemented left-to-right arithmetic, and it was possible to dispense with bracket levels. The range of functions was limited to the necessary and frequently used functions. These minimized problems arising from different designs and ensured simple and intuitive operation.

One example is the scientific calculator shown in the figure, developed and produced in Japan in 1975.

From o++o point of view, however, the TI-30 ECO RS behaves correctly for the most part in these problems. For example, with 2 to the power of 3 to the power of 4, it chooses the way of calculating that the majority of people prefer, namely to calculate from left to right. This is also true for engineers, as I experienced many times. That unary functions are typed after the number (the argument), we also welcome, because this way of calculation also follows the principle from-left-to-right:

3.14 sin cos

The calculator from Texas-Instruments first calculates the sine and applies then the cosine function to the result. This is not taught in math classes, but it is also easier to understand. Unfortunately, the calculator is not completely consistent at this point. At 1 + 2 × 3, it no longer calculates from-left-to-right. Now, it calculates as Descartes supposedly wanted it to do. Only so that one could write a polynomial somewhat more elegantly, humans gave up the general principle from-left-to-right to calculate. Since one can regard today also a list of numbers as input value, this argumentation from the 17-th century has no more right to exist from our point of view. Instead of

$X^3 + 2 * X^2 + 3 * X + 4$

we can today briefly and succinctly type: X poly 1 2 3 4 or X poly [1 2 3 4]

In general, we also estimate that most of today's calculators are morally worn out. They should no longer be used at school at all. The first electronic, actually palm-sized calculator was developed as early as 1967 and had - as is still common today - a **very small display**. Since cell phones with much larger displays exist today in 2023 and we also know much more powerful apps with a much wider range of applications, calculators should generally be banned from school today or displayed in the school museum.

Let's consider a very simple problem. You want to add 10 numbers with the Texas Instruments calculator. At the end of the calculation, when you realize that the result cannot be correct, you cannot look at the input again. You have to type in all the numbers again. It is unclear whether you do this correctly if all these numbers consist of 10 digits.

Let's continue by looking at the % key. If you play with the calculator and type for example

10 %

the result is 0.1.

So, you might suspect that the percent key is just mislabeled, and it just divides by 100. The percent key is also hard to type on this calculator, since you have to type 2nd beforehand. Also, once you find the little blue percent sign, which doesn't have its own key, you have to concentrate very hard to see if you should press the key above or below it. These are, of course, potential sources of error. Of course, you also have to know whether the 2nd key is only valid for the next operation or until I press it again. If one then types for example

10 + 10 %

If you press = , you first get 1. Only when you press = again does the current user get the number 11 that he probably wants.

But if you think mathematically, only one of the following two solutions comes into question:

10 + (10 %)

or

(10 + 10) %

You get 10.1 in the first case and 0.2 in the second.

That is, with this symbol mathematical thinking is contradicted. How should one understand

10 + 10 %

differently as a term? Why do all students need to learn a term definition if it is not applied in calculator practice at school?

To our knowledge, there is only one programming language that uses this symbol at all in connection with percentage calculation. Here, however, +% is used as a two-digit operation symbol. This also makes it mathematically clear and clean.

Just as the three letters of sin represent one operation symbol, +% is also one operation.

In o++o results in 10 +% 10 11. .

If you type in the Texas Instruments calculator

10 sin x^2

so you never see on the display which operation symbol you have just typed or typed before. Furthermore, the keyboard labels make it difficult to understand the "dot before dash" rule when the multiplication sign consists of 2 dashes and the division sign contains a dash. We conclude the section with what appears to be a very simple multiple addition. We think that hardly anyone can correctly manage an addition of very many numbers with a calculator.

7.8 Is EXCEL morally worn out?

Program 7.8.1: An o++o program for which EXCEL needs more than six worksheets!

```
<TAB!
NAME,    LENGTH,(AGE,WEIGHT m)m
Klaus    1.68    18  61
                 30  65
                 61  80
Rolf     1.78    40  72
Kathi    1.70    18  55
                 40  70
Walleri  1.00     3  16
Victoria 1.61    13  51
Bert     1.72    18  66
                 30  70
!TAB>
sel NAME! AGE>20
gib BMI,(AGE,BMI,(NAME,BMI m) m)  BMI:=WEIGHT:LENGTH:LENGTH!++:
rnd 2
```

If you realize this o++o program in EXCEL you need more than 6 worksheets. Hardly anyone overlooks these EXCEL sheets, which is why they are very difficult to change. More details can be found under o++o versus EXCEL. Spreadsheet programs have several advantages and are widely used, but they also have a number of disadvantages, which we will list:

1. Data and formulas are mixed. For this reason, and because an EXCEL worksheet can contain hundreds or even thousands of formulas, it is almost impossible to check the correctness of the programs or to adapt them to changes.
2. EXCEL does not know schemas for structured tables: e.g. SUBJECT,MARKl l describes a structured schema - here a list of subjects is described, and for each subject there is also a list of marks.
3. EXCEL can display structured tables visually, but it cannot sort them directly or process them reasonably.
4. You cannot use EXCEL to query databases, XML or Wikipedia. For that you would still have to learn SQL, XQuery or better o++o.
5. EXCEL formulas are relatively cryptic because, for example, they often contain individual cell designations. For example, the sum over a column is written in EXCEL in the form: =SUM(F12:F75)
6. A single EXCEL formula can require more analysis than a complete o++o program.
7. EXCEL contains only a few mathematical concepts and therefore requires an excessive amount of detailed knowledge.
8. EXCEL offers the comma as the decimal point to German and other users. However, this makes it difficult to exchange corresponding worksheets of international companies across country borders, since many countries prefer the decimal point.
9. Since data and programs are usually separated in o++o, the data can be used by several programs without any problems. This is more difficult with EXCEL.
10. For aggregations (sums, averages, maxima, etc.) per value you have in general to presort or group in EXCEL but not in o++o.
11. In EXCEL, you have to write each number in a separate cell. This could quickly overwhelm a smartphone screen.

12. o++o is based on an abstract tabment concept for data. A tabment can already be represented in many ways by default: web tab xml image column ... and also compact (hsq). With CSS, the output of o++o can be formatted almost arbitrarily. EXCEL, on the other hand, is based on a concrete print image. This makes it easier to create simple applications at first, but it is rather a disadvantage for the complexity of today's applications.
13. After studying the above criticisms of EXCEL, a VW engineer remarked: At VW, EXCEL can be used by any employee at will. As a rule, however, it is only simple tables that are to be made "nice". Sometimes a few simple arithmetic operations are necessary. Comprehensive, complex applications do not take place in EXCEL.

EXCEL does not know mass data operations and could be morally worn out for this reason alone. Therefore, I plead for removing EXCEL programs also from school lessons and replacing them by more powerful and promising concepts and systems.

7.9 o++o Proofs

Proofs have played a minor role in school and even outside the world of professional mathematicians. Yet everyone wants to have confidence in a calculation, a system, or a calculator. When confronted with a new type of calculator or system, everyone first tries to solve problems like `2 times 3`. Who suspects further problems, tests e.g. `1 plus 2 times 3`.

The highly respected German economist Professor Sinn says in his lecture **Energiewende ins Nichts** (see youtube) that calculations only really make sense if you can understand them. To do this, you have to understand all the sub-steps in detail.

We have been working on this requirement of Prof. Sinn for decades. The SQL designers had formulated this requirement somewhat differently at the beginning of their development:

SQL should become an end-user language.

It follows directly that the average consumer should be able to understand SQL programs. Today, however, almost all SQL programmers come from the computer science corner.

The importance of statistics in schools is increasing.

How to teach a student a new statistical function, such as the average `++:` or the function `mad` of o++o. If you simply apply the function to several lists of numbers and look at the result, you usually cannot understand its meaning. However, if the teacher knows that the students have already understood the functions `++` (sum) and `++1` (count), this is no longer so difficult.

Program 7.9.1: Preparation of an o++o proof for the ++: operation..

```
Xl:= 3 5 4 2 1
SUM:=Xl ++
CNT:=Xl ++1
MYAVG:=SUM:CNT
OTTOAVG:=Xl++:
```

Result (ment)

```
<TABM>
  <SUM>15</SUM>
  <COUNT>5</COUNT>
  <AVERAGE>3.</AVERAGE>
  <OTTOAVG>3.</OTTOAVG>
  <X>3</X>
  <X>5</X>
  <X>4</X>
  <X>2</X>
  <X>1</X>
</TABM>
```

Despite this (docu)ment output, it is clear that the two average values match. Here the ment output agrees almost completely with the xml output. The columns MYAVG and OTTOAVG must however agree with all other input lists. The program has the advantage of being very simple. But the student still has to enter a lot of data. Using the example of o++o-mad, which has not yet played a big role in Germany, we want to show that an extended o++o program can relieve us of much of the typing work. This mad function is one of the simplest and clearest statistical functions, but it has not so nice mathematical properties. Now, we assume knowledge of the operations `++:`, `..x` and `abs`. By `from ..x to!cnt`
a list of `cnt` random numbers between `from` and `to` is generated. `abs` calculates the absolute value.

Program 7.9.2: o++o Proof for the ++: Operation.

```
RANDOMNRl:= 1 ..x 10!10
Xl:= 1 ..x RANDOMNR!RANDOMNR
AVG:=Xl ++:
DISTANCE:=AVG - X abs
MYMAD:=DISTANCEl ++:
OTTOMAD:=Xl mad
gib AVG,MYMAD,OTTOMAD l
```

Result (tab)

```
AVG,          MYMAD,         OTTOMAD
2.8           1.44           1.44
4.16666666667 1.5            1.5
1.33333333333 0.444444444444 0.444444444444
1.5           0.5            0.5
5.            2.8            2.8
1.            0.             0.
3.            1.6            1.6
1.            0.             0.
1.            0.             0.
1.            0.             0.
```

We can easily extend the result table to a thousand output rows table by replacing the last number 10 of the first row with 1000. We have extracted only the relevant columns with the gib statement.

7.10 An example of deep digitization

Perhaps the following example makes the concept of deep digitization a little clearer: If addition, multiplication, **were not** taught in school, today, for example, you would need different apps to solve the following two problems.

An analogy to deep digitization from the field of "single data" operations

1. I have received a load of 36.57 tons of bulk material and will receive 31 more loads of this type. How much bulk will I have in total?
2. I have a rectangular plot of land 32 m wide and 36.57 m long. What is the size of my plot?

Everyone who has understood multiplication knows that it is one and the same problem that can be solved very easily with a simple calculator. For today's digitization, this means that a deep digitization could require far fewer computer applications than a FD (flat digitization) and that the end users (managers, politicians, ...) could master far more traditional applications (apps). After all, if the apps and applications are based on one (e.g. o++o) data model, one can of course also standardize the interfaces of these apps and such an application could replace many conventional FD applications.

8 Schemes and Structured Tables

All column names of a table are often considered as a schema of the table. Column names are necessary to understand corresponding column values correctly. If we consider structured tables, it is advantageous to enrich the column names with corresponding collection symbols; for example, l for list.

NAME,	BORNIN,	(DEED,	YEAR l) l
Otto the Great	Old Saxony(De)	Elected King of Germany	936
		The Hungarians defeated on the Lechfeld	955
		First emperor of the Holy Roman Empire	962
Otto von Moravia	Moravia	married Euphemia of Hungary	1086
Otto von Guericke	MD(De)	Inventor of the air pump	1649
		Hemisphere test for the emperor	1654
Otto von Bismarck	Preussen(De)	with carrot and stick policy	1871
		Ems Dispatch	1870
		First Chancellor of Germany	1871
Nicolaus Otto	Taunus (De)	Co-inventor of the gasoline engine	1876
OttoNormalVerbraucher	De	learns car driving	1960
		learns a programming language	2025

Tabment 8.1: ottos.tab

The above table (TABMENT=TABelle+dokuMENT) ottos.tab contains a list of 6 "persons" and for each person a repeating group (DEED, YEAR l) - a list of (DEED, YEAR)-pairs. Here, a person has 4 columns, but it is a triple (3-tuple). It is a structured tuple, struple for short (designation of Prof. Schek). The first two components are of type TEXT and the third component is a list of sub tuples (pairs) (2-tuples). We will call the attribute values of a level segment.

The first NAME segment is:

NAME,	BORNIN
Otto the Great	Old Saxony(De)

The first DEED segment of Otto the Great reads:

DEED,	YEAR
Elected King of Germany	936

The first person corresponds to the first struple; it is a NAME tuple:

NAME,	BORN,	(DEED,	YEAR l)
Otto the Great	Old Saxony(De)	Elected King of Germany	936
		The Hungarians defeated on the Lechfeld	955
		First emperor of the Holy Roman Empire	962

Since the DEED tuples (= DEED sub-tuples) do not contain any other collections, a DEED segment is the same as a DEED tuple. If we were to represent the above table by an ordinary flat table, every (NAME, BORNIN)-pair would have to appear in every row. That is, (Otto the Great, Old Saxony(De)) would have to appear 3 times (once for each DEED segment). Then, for example, it is not so easy to count the persons in the table. With the above table, the corresponding program looks like this:

Program 8.1: How many ottos are contained in the table ? (How many elements (struples) does the outermost collection contain?)

```
ottos.tab
++1
```

Here and in the following, we use these abbreviations and keywords:

aus: from

++1: count

gib: (corresponds to the SELECT of SQL)

sel : selection

sel-: selection

:= : assignment (extends the specified table by a new (complex) column)

m: set: contains different elements

b: Bag: an element may occur more than once

l: list: the order of the elements is important

The result of program 8.1 is a simple table:

```
ZAHL
6
```

The schema of this table does not contain a collection symbol because the table contains exactly one element. Similarly, we do not need a collection symbol in the following 2 queries. We do not want to explain the following queries in detail. We use the queries to illustrate what different types of tables there are and what schemas belong to them.

Program 8.2: How many persons and how many deeds are contained in the file ottos.tab?

```
ottos.tab
gib CNTPERSON,CNTDEED
    CNTPERSON:= NAME! ++1
    CNTDEED  := DEED! ++1
```

Result (tab)

```
CNTPERSON, CNTDEED
6          12
```

Program 8.3: Tell me the name of the person born in Saxony.

```
aus ottos.tab
sel Saxony in BORNIN
gib NAME
```

Result

```
NAME
Otto the Great
```

Program 8.4: Give me the name of a noble person.

```
aus  ottos.tab
sel  von in NAME
gib  NAME
```

Result

```
NAME
Otto von Moravia
```

If keywords like "sel " and "in" are highlighted in color in the future o++o software, the second program line will also be easier to read.

Here it would be better to output the names of all the nobles:

Program 8.5: Sort all noble names

```
aus  ottos.tab
sel  von in NAME
gib  NAMEm
```

Result (tab)

```
NAMEl
Otto von Bismarck
Otto von Guericke
Otto von Moravia
```

To save space on the screen or paper, we can also arrange the elements of a one-column-list or other collection horizontally:

Result (tabh)

```
NAMEl
"Otto von Bismarck" "Otto von Guericke" "Otto von Moravia"
```

Program 8.6: Count all deeds and all deeds of each century. Add to each century the corresponding people.

```
aus ottos.tab
CENTURY:=YEAR div 100 +1
gib CNTDEED,(CENTURY,CNTDEED,NAMEm m)
    CNTDEED:= DEED ! ++1
```

Result (Table with 3 segment types: CNTDEED, (CENTURY, CNTDEED2) and NAME)

```
CNTDEED,(CENTURY,CNTDEED2,NAMEm m)
12       10      3        Otto the Great
         11      1        Otto von Moravia
         17      2        Otto von Guericke
         19      4        Nicolaus Otto
                          Otto von Bismarck
         20      1        John Doe
         21      1        John Doe
```

Program 8.7: Count all the acts and the acts of each century with corresponding persons, where for each act the corresponding person must appear (with duplicates).

```
aus ottos.tab
CENTURY:=YEAR div 100 +1
gib CNTDEED,(CENTURY,CNTDEED,NAMEb m)
    CNTDEED:= DEED ! ++1
```

Result (tab)

```
CNTDEED,(CENTURY,CNTDEED2, NAMEb m)
12       10      3         Otto the Great
                           Otto the Great
                           Otto the Great
         11      1         Otto von Moravia
         17      2         Otto von Guericke
                           Otto von Guericke
         19      4         Nicolaus Otto
                           Otto von Bismarck
                           Otto von Bismarck
                           Otto von Bismarck
```

20	1	OttoNormalVerbraucher
21	1	OttoNormalVerbraucher

Where b stands for bag (multiset). So far, we have considered only tables with nested levels. But a structured table may also contain "independent" collections:

NAME ,	RESIDENCEl,	WIFEl,	RULESOVERl l
Otto the Great	Magdeburg	Editha	Saxony
	Memleben	Adelheid	Thuringia
			Bavaria
			Franconia
			Swabia
			Italy
			Bohemia
			Holland
			Lorraine
			Friesland
Charles IV	Prague	Margaret	Bohemia
	Tangermünde	Anna	Silesia
		Anna	Brandenburg
		Elizabeth	Italy
			Hungary

Tabment 8.2: emperors.tab

In this table "Memleben" and "Otto the Great" are in the same relation to each other as "Adelheid" and "Otto the Great". But this does not mean that "Adelheid" and "Memleben" are related to each other although they are in the same row. Therefore, the following restructuring is senseless.

Program 8.8: Query with empty result

```
aus emperors.tab
gib NAME,RESIDENCE,WIFE m
```

gib NAME,RESIDENCEm,WIFEm m is useful, however.

9 Tabment types (TTs) and structured documents

For structured tables and documents, we use the name Tabment. Therefore, we abbreviate the type of a tabment with TT (Tabment Type). The TT completes the information given by a schema. It specifies for each tag its schema. For example, the TT for the above table ottos.tab is:

```
TABMENT! OTTOS
OTTOS! NAME,BORNIN,(DEED,YEAR l)l
NAME BORNIN DEED! TEXT
YEAR! ZAHL
```

TEXT and ZAHL (integer) are elementary types that need no further explanation. Each named tabment is surrounded by a tag that is derived from the file name by omitting the type suffix. Therefore, our first table can also be presented in document style or in a document style with inner tables (ment or xml).

for example:

```
<OTTOS>
  <NAME>Otto the Great</NAME>
  <BORNIN>Altsaxony(De)</BORNIN>
  <DEED>Elected  King of Germany</DEED>.
  <YEAR>936</YEAR>
  <DEED>Hungarians beaten on the Lechfeld</DEED>.
  <YEAR>955</YEAR>
  <DEED>First emperor of the Holy Roman Empire</DEED>
  <YEAR>962</YEAR>
  <NAME>Otto von Moravia</NAME>
  <BORNIN>Moravia</BORNIN>
  <DEED>married Euphemia of Hungary</DEED>
  <YEAR>1086</YEAR>
  <NAME>Otto von Guericke</NAME>
  <BORNIN>MD (De)</BORNIN>
  <DEED>Inventor of the air pump</DEED>.
  <YEAR>1649</YEAR>
  <DEED>Half ball attempt in front of the emperor</DEED>.
  <YEAR>1654</YEAR>
  <NAME>Otto von Bismarck</NAME>
  <BORNIN>Preussen(De)</BORNIN>
  <AT>with carrot and stick policy</DEED>.
  <YEAR>1871</YEAR>
  <DEED>Ems Dispatch</DEED>.
  <YEAR>1870</YEAR>
  <AT>First Chancellor of the Reich of Germany</DEED>.
  <YEAR>1871</YEAR>
  <NAME>Nicolaus Otto</NAME>
  <BORNIN>Taunus (De)</BORNIN>
  <DEED>Miter inventor of the gasoline engine</DEED>.
  <YEAR>1876</YEAR>
  <NAME>Otto Normal Consumer</NAME>
  <BORNIN>De</BORNIN>
  <DEED>learns to drive</DEED>
  <YEAR>1960</YEAR>
  <DEED>learns a programming language</DEED>.
  <YEAR>2025</YEAR>
</OTTOS>
```

Tabment 9.1: Table ottos.tab in XML document style

```
"Otto the Great" Old Saxony(De)
 "Elected King of Germany" 936
 "The Hungarians defeated on the Lechfeld" 955
 "First Emperor of the Holy Roman Empire" 962
"Otto of Moravia" Moravia
 "married Euphemia of Hungary" 1086
"Otto von Guericke" "MD (De)"
 "Inventor of the air pump" 1649
 "Hemisphere trial before the emperor" 1654
"Otto von Bismarck" Prussia(De)
 "with carrot and stick policy" 1871
 "Ems Dispatch" 1870
 "First Imperial Chancellor of Germany" 1871
"Nicolaus Otto" "Taunus (De)"
 "Co-inventor of the gasoline engine" 1876
"Otto Normal Consumer" De
 "learns to drive" 1960
 "learns a programming language" 2025
```
Tabment 9.2: Table ottos.tab in hsq style

We consider a document with TT. It uses alternatives through (|). It comes from the XQuery use cases (C+07).

```
<META!
TABMENT! REPORT1
REPORT1! SECTIONl
SECTION! TITLE,CONTENT
CONTENT! TEXT|NARCOSIS|PREPARATION|CUT|ACTION|OBSERVATION l
PREPARATION! TEXT|ACTION l
CUT! TEXT|GEOGRAPHY|INSTRUMENT l
ACTION! TEXT|INSTRUMENT l
TITLE NARCOSIS OBSERVATION GEOGRAPHY INSTRUMENT! TEXT
!META>
<REPORT1>
  <SECTION>
    <TITLE>Procedure</TITLE>
    <CONTENT>
  The patient was taken to the operating room, where she was placed in the supine position and
<NARCOSIS> induced under general anesthesia. </NARCOSIS>
<PREPARATION>
<ACTION>A Foley catheter was placed to decompress the bladder</ACTION> and the abdomen was then
sterilely prepped and draped.
</PREPARATION>
<CUT>
A curved incision was made
<GEOGRAPHY> in the center line immediately infraumbilical </GEOGRAPHY>
 and the subcutaneous tissue was divided
 <INSTRUMENT> Use electrocautery. </INSTRUMENT>
 </CUT>
 The fascia was identified and
 <ACTION> # 2 0 Maxon seams were placed on each side of the centerline.
 </ACTION>
 <CUT>
 The fascia was shared with
<INSTRUMENT> electrocautery </INSTRUMENT>
 and the peritoneum entered.
</CUT>
 <OBSERVATION>The small intestine was identified.</OBSERVATION>
 and
 <ACTION> the <INSTRUMENT>Hasson trocar</INSTRUMENT>
 was placed under direct visualization.
 </ACTION>
 <ACTION>The <INSTRUMENT>Trocar</INSTRUMENT>using the
      Sutures was attached to the fascia.
</ACTION>
</CONTENT>
</SECTION>
```

```
</REPORT1>
```

Tabment 9.3: report1.ment

In report1.xml the CONTENT is a list of elements, where each element is either of type TEXT, ANESTESIA, PREPARATION, CUT, ACTION or OBSERVATION. In the above document, the first element is simple TEXT, the second is of type ANESTESIA, the third is of type PREPARATION, etc. Since our report was tagged in the above way, the following example queries are possible.

For example:

Program 9.1: What instruments were used in the second cut?

```
aus report1.ment
gib CUTl
sel CUT pos = 2
gib INSTRUMENTl
```

```
Result (tab)
```

```
INSTRUMENTl
```

```
electrocautery
```

Program 9.2: What are the first two instruments used?

```
aus report1.ment
gib INSTRUMENTl
sel INSTRUMENT pos < 3
```

```
Result (tab)
```

```
INSTRUMENTl
```

```
Use electrocautery.
electrocautery
```

10 A university database

We consider a non-relational database consisting of one flat and two structured tables:

```
FACS! FAC,DEAN,BUDGET,STUDCAPACITY m
STUDENTS! STID,NAME,LOC?,STIP,FAC,(COURSE,MARK m),(PROJ,HOURS m) m
COURSES! COURSE,TEACHER,(ISBN,TITLE m)m
```

The underlined column names are keys. The last two tables can be represented by the following 5 flat relations:

student1:	STID,NAME,LOC?,STIP,FAC m
exam1:	STID,COURSE,MARK m
projects1:	STID,PROJ,HOURS m
course1:	COURSE,TEACHER m
course_books1:	COURSE,ISBN,TITLE m

FAC,	DEAN,	BUDGET,	STUDCAPACITY m
Art	Sitte	2'000	600
Infor	Reichel	10'000	500
Math	Dassow	1'000	200
Philo	Hegel	1'000	10
Sport	Streich	8'000	150

Tabment 10.1: facs.tab

STID,	NAME,	LOC?,	STIP,	FAC,	(COURSE,	MARK m),	(PROJ,	HOURS m)m
1111	Ernst	Oehna	500	Math	Algebra	1	Fritz	4
					Logic	2	Otto	2
					History	1		
2222	Sophia	Berlin	400	Infor	Algebra	3	Ghandi	5
					Databases	1	Ming	4
					Otto	1	Otto	6
3333	Clara	Oehna	450	Infor	Databases	1		
					OCaml	2		
4444	Ulrike		400	Art			Monet	10
5555	Käthe	Gerwisch	600	Art	Repin	1	Monet	20
					Apel	1		
6666	Claudia	Berlin	600	Sport	Psycho	2	Matthes	8
					Ski	1	Witt	12

Tabment 10.2: students.tab

COURSE,	TEACHER,	(ISBN,	TITLE m)m
Algebra	Reichel	0138-3019	Structural Induction on Partial Alg.
		3-8244-2099-6	Structured tables
Databases	Saake	0-321-31256-2	Database Systems an Application
		0-7167-8069-0	Principles of Database Systems
Otto	Benecke	0-7167-8069-0	Principles of Database Systems
		3-8244-2099-6	Structured tables

Tabment 10.3: courses.tab

STID,	NAME,	LOC?,	STIP,	FAC m
1111	Ernst	Oehna	500	Math
2222	Sophia	Berlin	400	Infor
3333	Clara	Oehna	450	Infor
4444	Ulrike		400	Art
5555	Käthe	Gerwisch	600	Art
6666	Claudia	Berlin	600	Sport

Tabment 10.4: students1.tab

```
STID,COURSE,   MARK m
1111 Algebra   1
1111 History   1
1111 Logic     2
2222 Algebra   3
2222 Databases 1
2222 Otto      1
3333 Databases 1
3333 OCaml     2
5555 Apel      1
5555 Repin     1
6666 Psycho    2
6666 Ski       1
```
Tabment 10.5: exams1.tab

```
STID,PROJ,   HOURS m
1111 Fritz    4
1111 Otto     2
2222 Ghandi   5
2222 Ming     4
2222 Otto     6
4444 Monet   10
5555 Monet   20
6666 Matthes  8
6666 Witt    12
```
Tabment 10.6: projects1.tab

The above tables and the following programs refer to tab files, although we keep in mind that the specified tables could be database tables.

10.1 Selection (sel sel-)

A condition specifies tuples or sub tuples. In a *sel* clause the specified tuples form the result, in a *sel-* clause the specified tuples are omitted.

Consequently, the schema and the TT of the considered tabment are not changed by a selection. Column names or tags are written in upper case in an o++o program. They must start with a letter or the character "_". A WORT (word) that is not enclosed by "-symbols must therefore use a lowercase letter. TEXT may contain spaces; however, they must then be enclosed in "-symbols.

Program 10.1.1: Find all students from Berlin and Oehna with bad results..

```
aus students.tab
sel LOC in "Berlin Oehna" # selects students
sel MARK > 2              # selects exams and students
```
Result (tab)

```
STID,NAME , LOC?  ,STIP,FAC ,(COURSE ,MARK m),  (PROJ , HOURS m) m
2222 Sophia Berlin 400  Infor Algebra 3          Ghandi  5
                                                 Ming    4
                                                 Otto    6
```
Intermediate result after the first condition

```
1111 Ernst  Oehna  500 Math      Algebra    1     Fritz  4
                                 History    1     Otto   2
                                 Logic      2
2222 Sophia Berlin 400 Infor     Algebra    3     Ghandi 5
                                 Databases  1     Ming   4
```

```
                                Otto      1     Otto    6
3333 Clara   Oehna  450 Infor   Databases 1
                                OCaml     2
6666 Claudia Berlin 600 Sport   Psycho    2     Matthes  8
                                Ski       1     Witt    12
```

The second "condition" is applied to the result of the first condition. The second "condition" is an abbreviation for the following two conditions:

```
sel STID!   MARK>2 # Selection STID tuple (MARK>2 must exist)
sel COURSE! MARK>2 # Selection COURSE tuple
```

The first of these two conditions expresses that we select (complete) student tuples for which there exist (COURSE,MARK) sub tuples with a grade of 3 or higher. We do not write the existence quantifier because there are several EXIST quantifier behind each condition. "#" is the comment symbol for a line. It can be used to describe the meaning of a program step. Also, lines can be commented out to indicate intermediate results.

Program 10.1.2: For all students from Oehna and Berlin, indicate all results with 3 or worse.

```
aus students.tab
sel LOC in "Berlin Oehna" # equivalent: LOC in [Berlin Oehna]
                          # equivalent: LOC in Berlin Oehna
sel COURSE! MARK>2        # selects exams and not students
gib NAME,LOC,(COURSE,MARK m) b
```

Result (tab)

```
NAME ,  LOC , (COURSE, MARK m) b
Clara   Oehna
Claudia Berlin
Ernst   Oehna
Sophia  Berlin Algebra 3
```

After applying the two conditions, the restructuring (see section 10.3) was applied. Therefore, the scheme of the result has changed and the data has been sorted.

Program 10.1.3: Find all students from Oehna and Berlin with a grade of 3 or worse, with all marks.

```
aus students.tab
sel LOC in "Berlin Oehna"
sel STID! MARK>2                # selects only students and not exams
gib NAME,LOC,(COURSE,MARK m)m
```

Result (tab)

```
NAME , LOC ,  (COURSE,   MARK m) m
Sophia Berlin  Algebra    3
               Databases 1
               Otto      1
```

Program 10.1.4: Find all students who have only a grade of 1 and at least one grade of 1.

```
aus students.tab
sel MARKm = {1}    # { } are set brackets
```

Result (tab)

```
STID,NAME , LOC?    ,STIP,FAC ,(COURSE,MARK m),(PROJ ,HOURS m) m
5555 Käthe  Gerwisch 600  Art    Apel    1         Monet 20
                                 Repin   1
```

For the evaluation of the condition, for each student the list of his marks is transformed into a set. Thus, Ernst's set {1 2 1} = {1 2} and Kathe's set {1 1} is equal to {1}. Two sets are equal if every element of the left side is also on the right side and every element of the right side is on the left side. In other words, two sets M1 and M2 are equal if 'M1 inmath M2 & M2 inmath M1' holds. If we want to have all students with exactly two marks 1, then we can use multisets: MARKb = {{1 1}} (b abbreviates Bag). If the order of the marks is also important, then we can take lists: MARKl = [1 2 1], etc.

Program 10.1.5: Find all students with all exams, who got a 1 in the algebra course..

```
aus  students.tab
sel  STID! COURSE=Algebra & MARK=1
gib  STID,NAME,(COURSE,MARK m)m
```

Result (tab)

```
STID,NAME ,(COURSE, MARK m) m
1111 Ernst  Algebra 1
            History 1
            Logic   2
```

Program 10.1.6: Find all students with all exams, who have taken an algebra course and have a 1 (not necessarily in the same course)..

```
aus students.tab
sel STID! COURSE=Algebra
sel STID! MARK=1
gib STID,NAME,(COURSE,MARK m)m
```

Result (tab)

```
STID,NAME , (COURSE ,  MARK m) m
1111 Ernst   Algebra   1
             History   1
             Logic     2
2222 Sophia  Algebra   3
             Databases 1
             Otto      1
```

Program 10.1.7: Find all students who already have exams in Algebra and Databases..

```
aus students.tab
sel STID! COURSE=Algebra
sel STID! COURSE=Databases
#sel Algebra Databases in COURSEm is equivalent to both selections
```

Result (tab)

```
STID,NAME , LOC?  ,STIP,FAC ,  (COURSE ,  MARK m),(PROJ , HOURS m) m
2222 Sophia Berlin 400  Infor    Algebra    3         Ghandi 5
                                 Databases 1         Ming   4
                                 Otto       1         Otto   6
```

Intermediate result after the first condition

```
1111 Ernst  Oehna   500  Math    Algebra    1         Fritz  4
                                 History    1         Otto   2
                                 Logic      2
```

```
2222 Sophia Berlin 400  Infor   Algebra    3         Ghandi 5
                                Databases 1         Ming   4
                                Otto       1         Otto   6
```

If we would connect both conditions by & (and), this condition "contains" only one EXIST quantifier, of the kind that no sub tuple exists that satisfies both sub conditions simultaneously. The result would be empty in any case.

Program 10.1.8: For each student who has completed Algebra, indicate all other courses they have completed.

```
aus  students.tab
sel  STID!   COURSE=Algebra     # selects students
sel- COURSE! COURSE=Algebra     # chooses exams
gib  NAME,COURSEb m
```

Result (tabh)

```
NAME,   COURSEb m
Ernst   History Logic
Sophia  Databases Otto
```

Program 10.1.9: Find all occurrences of the word Otto.

```
aus students.tab
sel Otto
```

Result (tab)

```
STID,NAME,    LOC?  ,STIP,FAC ,  (COURSE ,MARK m),(PROJ ,HOURS m) m
1234  Ernst  Oehna  500  Maths                    Otto  2
1245  Sophia Berlin 400  Infor   Otto     1       Otto  6
```

Program 10.1.10: Print from all tuples of the university database (to which I have access) the (sub)tuples containing the word Apel.

```
aus students.tab,courses.tab
sel Apel
```

Result (xml)

```
<TABM>
  <STUDENTS>
    <STID>5555</STID>
    <NAME>Käthe</NAME>
    <STIP>600</STIP>
    <FAC>Art</FAC>
    <COURSE>Apel</COURSE>
    <MARK>1</MARK>
  </STUDENTS>
  <COURSES/>
</TABM>
```

So far in this section we have only considered "selection by content", but almost the same importance has "selection by position". This is not only useful for lists, but can also be used in the context of "relational applications". We only consider two examples here.

Program 10.1.11: Give for each student from Oehna with exams, the last exam.

```
aus students.tab
sel LOC=Oehna
sel MARK pos- = 1
gib STID,NAME,(COURSE,MARK m)m
```

Result (tab)

```
STID,NAME,(COURSE,MARK m) m
1111 Ernst Logic  2
3333 Clara OCaml  2
```

The pos (pos-) function returns the position number (position number backwards) of the (sub-) item in the corresponding set. Therefore, MARK pos is the same as COURSE pos.

Program 10.1.12: Give the 2 best exams for the 3 best students. We omit Ulrike because we cannot calculate an average for her. She has no marks yet.

```
aus  students.tab
sel- NAME=Ulrike
sel  MARK=MARK
gib  AVGM,NAME,FAC,(MARK,COURSE m)m
     AVGM:= MARK! ++:
sel  NAME pos < 4
sel  MARK pos < 3
rnd  2
```

Result (tab)

```
AVGM,NAME ,FAC ,  (MARK,COURSE m ) m
1.00 Käthe Art     1    Apel
                   1    Repin
1.33 Ernst Math    1    Algebra
                   1    History
1.50 Clara Infor   1    Databases
                   2    OCaml
```

Here, it is sufficient to know that by the gib-clause the students are sorted by AVGM, the exams are sorted by MARK and AVGM is the average for each student. The gib clause is explained in more detail in section 10.3.

Although the following query does not require *sel* or *sel-*, *first* and *last* are still selections. *last* selects the last element from each collection. These operations can be used to quickly get a first impression of the structure and content of a tabment.

Program 10.1.13: Find the last element of each deepest collection of the students file.

```
exams1.tab,projects1.tab last
```

Result (tab)

```
STID ,COURSE ,MARK m, (STID ,PROJ ,HOURS  m)
6666  Ski      1        6666  Witt  12
```

Program 10.1.14: Find the students with the highest scholarships.

```
aus students.tab
STIPMAX:=STIPl max        # a new column with one value is created
sel STIP=STIPMAX
```

Result (tab)

```
STIPMAX ,(STID ,NAME    ,LOC?    ,STIP ,FAC ,(COURSE ,MARK  m),(PROJ   ,HOURS  m) m)
600       5555  Käthe   Gerwisch 600   Art   Apel    1         Monet    20
                                             Repin   1
          6666  Claudia Berlin   600   Sport Psycho  2         Matthes  8
                                             Ski     1         Witt     12
```

10.2 Calculations (:=)

Program 10.2.1: Calculate the gross values of several prices.

```
3.18 55.88 17.90 * 1.19
```

Result (hsqh and tabh)

```
PZAHLl
3.7842 66.4972 21.301
```

Program 10.2.2: Calculate the gross values of several prices and leave the entered values in the output.

```
NETl :=3.18 55.88 17.90
GROSS:=NET * 1.19
```

Result (tab)

```
NET,  GROSS l
 3.18  3.7842
55.88 66.4972
17.9  21.301
```

Program 10.2.3: Calculate the gross values of several prices.

```
3.18 55.88 17.90 +% 19
```

Result (tabh)

```
PZAHLl
3.7842 66.4972 21.301
```

Program 10.2.4: Convert all net prices of a small table into gross prices..

```
<TAB!
ARTICLE,  PRICE l
OttoRAMDB 500
OttoWiki   10
OttoCalc   20
!TAB>
+% 19
```

Result

```
ARTICLE , PRICE l
OttoRAMDB 595.
OttoWiki   11.9
OttoCalc   23.8
```

19% is added to each number in the table. Text values are not changed by arithmetic operations with numbers.

Program 10.2.5: Calculate the gross value of each item and the sum of all gross values.

```
<TAB!
ARTICLE,  PRICE,CNT m
OttoRAMDB 500      20
OttoWiki   10     200
OttoCalc   20    4000
!TAB>
TOTAL:=PRICE*CNT +% 19
TOTALSUM:=TOTALl ++
```

Result

```
TOTALSUM,(ARTICLE ,PRICE,CNT, TOTAL m)
 109480.  OttoCalc   20  4000 95200.
          OttoRAMDB 500    20 11900.
          OttoWiki   10   200 2380.
```

Program 10.2.6: Report each computer science student's stipend in dollars.

```
aus  students.tab
sel  FAC=Infor
DOL:=STIP*1.02
```

Result (tab)

```
STID,NAME , LOC?  ,STIP,FAC , DOL , (COURSE    , MARK m),(PROJ , HOURS m) m
2222 Sophia Berlin 400  Infor 408.   Algebra    3        Ghandi 5
                                     Databases  1        Ming   4
                                     Otto       1        Otto   6
3333 Clara  Oehna  450  Infor 459.   Databases  1
                                     OCaml      2
```

Program 10.2.7: Pay each student 100 euros for each of their projects..

```
aus students.tab
BONUS:= PROJl ++1 *100
gib STID,NAME,BONUS m
```

Result (tab)

```
STID,NAME,   BONUS m
1111 Ernst   200
2222 Sophia  300
3333 Clara     0
4444 Ulrike  100
5555 Käthe   100
6666 Claudia 200
```

Program 10.2.8: Pay each Oehna student an additional bonus based on their mark average.

```
aus students.tab
sel LOC=Oehna
AVG1:= MARKl ++:
BONUS3:=1000 : AVG1
gib STID,NAME,AVG1,BONUS3 m
rnd 2
```

Result (tab)

```
STID,NAME ,AVG1,BONUS3 m
1111 Ernst 1.33 750.00
3333 Clara 1.50 666.67
```

With the help of rnd (round) every value of a table is rounded to 2 digits after point (dot). For texts the value remains unchanged again.

Program 10.2.9: The students of the math faculty get a bonus of 900 euros, the computer science of 800 euros and all others get 700 euros.

```
aus students.tab
BONUS:= 900 if FAC=Math !
        800 if FAC=Infor!
        700
```

```
gib STID,NAME,FAC, BONUS m
```
Result (tab)
```
STID, NAME ,  FAC , BONUS m
1111  Ernst   Math  900
2222  Sophia  Infor 800
3333  Clara   Infor 800
4444  Ulrike  Art   700
5555  Käthe   Art   700
6666  Claudia Sport 700
```

Program 10.2.10: Calculate the BMI (body mass index) for each weight of each person
```
<TAB!
NAME, LENGTH, (AGE, WEIGHT l)l
Klaus 1.68      18   61
                30   65
                56   80
                61   75
Kathi 1.70      18   55
                40   70
!TAB>
BMI:= WEIGHT : LENGTH : LENGTH
rnd 2
```
Result (tab)
```
NAME ,LENGTH, (AGE, WEIGHT, BMI l) l
Klaus 1.68      18   61       21.61
                30   65       23.03
                56   80       28.34
                61   75       26.57
Kathi 1.70      18   55       19.03
                40   70       24.22
```

10.3 Restructuring (gib)

The restructuring operation stroke allows to restructure any tabment into another arbitrary tabment only by specifying the scheme or the TT of the target tabment. Additionally, aggregations, elimination of duplicates, union, sorting and certain joins can be realized.

Program 10.3.1: Illustrate the collection symbols
```
aus students.tab
gib FACm,FACb,FACl,FACm-,FACb-,FACl-,FAC?
```
Result (tab)
```
FACm ,FACb ,FACl ,FACm- ,FACb- ,FACl- ,FAC?
Art   Art   Math  Sport  Sport  Sport  Math
Infor Art   Infor Math   Math   Art
Math  Infor Infor Infor  Infor  Art
Sport Infor Art   Art    Infor  Infor
      Math  Art          Art    Infor
      Sport Sport        Art    Math
```
The STID-segments (type: (STID, NAME, LOCATION?, STIP, FAC)) are inserted one after another into each of the given FAC collections. COURSE and PROJ segments are ignored.

Program 10.3.2: Sort students by FAC and NAME.

```
aus students.tab
gib FAC,NAMEb m
```

Result (tabh)

```
FAC,  NAMEb m
Art   Käthe
      Ulrike
Infor Clara
      Sophia
Math  Ernst
Sport Claudia
```

STID segments are inserted first into the FAC level and then deeper into the NAME level, segment by segment. COURSE and PROJ segments are not touched anymore.

Program 10.3.3: Sort students by FAC and NAME, resulting in a flat table.

```
aus students.tab
gib FAC,NAME m
```

Result (tab)

```
FAC , NAME m
Art   Käthe
Art   Ulrike
Infor Clara
Infor Sophia
Math  Ernst
Sport Claudia
```

If we replace m with b, the result elements do not change.

Program 10.3.4: Sort the faculties downwards by BUDGET and secondly by student capacity.

```
aus facs.tab
gib BUDGET,STUDCAPACITY,FAC m-
```

Result (tab)

```
BUDGET,STUDCAPACITY,FAC m-
 10000 500          Infor
  8000 150          Sport
  2000 600          Art
  1000 200          Math
  1000 10           Philo
```

Program 10.3.5: Sort the faculties by budget and additionally by student capacity. (two independent sortings of one table).

```
aus facs.tab
gib BUDGET,FAC m-,(STUDCAPACITY,FAC m-)
```

Result (tab)

```
BUDGET,FAC m-,(STUDCAPACITY,FAC m-)
10000  Infor   600          Art
 8000  Sport   500          Infor
 2000  Art     200          Math
 1000  Philo   150          Sport
 1000  Math     10          Philo
```

Program 10.3.6: Pack each student's exam data by department. (Re-group already grouped data).

```
aus students.tab
gib FAC,(COURSE,MARK b)m
```

Result (tab)

```
FAC ,(COURSE ,  MARK b) m
Art   Apel      1
      Repin     1
Infor Algebra   3
      Databases 1
      Databases 1
      OCaml     2
      Otto      1
Math  Algebra   1
      History   1
      Logic     2
Sport Psycho    2
      Ski       1
```

Here the STID segments are inserted into the FAC level. They cannot be inserted deeper because they contain neither COURSE nor MARK values. The corresponding exams bags are then initially always empty. Then each COURSE segment ((COURSE, MARK)-pair) is extended by its parent STID segment. These extended segments can be inserted step by step into the corresponding bags (b). The extended segment has the type: (STID, NAME, LOC?, STIP, FAC, COURSE, MARK). PROJ-segments are not needed.

Program 10.3.7: (Special selection with gib clause) Give all students, for which a LOC entry exists, with this entry. Give additionally the given collection for comparison purposes.

```
aus students.tab
gib NAME,LOC m,(NAME,LOC? m)
```

Result (tab)

```
NAME ,  LOC m,  (NAME ,  LOC? m)
Clara   Oehna    Clara   Oehna
Claudia Berlin   Claudia Berlin
Ernst   Oehna    Ernst   Oehna
Käthe   Gerwisch Käthe   Gerwisch
Sophia  Berlin   Sophia  Berlin
                 Ulrike
```

In the first set, the user requests complete pairs. Since no pair exists for Ulrike, she cannot appear in the first result.

Program 10.3.8: (Selection with gib clause only.) Specify all students with non-empty exam-collections.

```
aus students.tab
gib STID,NAME,FAC,COURSE,MARK m
gib STID,NAME,FAC,(COURSE,MARK m)m
```

Result (tab)

```
STID, NAME ,  FAC ,(COURSE ,  MARK m) m
1111  Ernst   Math  Algebra   1
                    History   1
                    Logic     2
2222  Sophia  Infor Algebra   3
                    Databases 1
                    Otto      1
```

```
3333  Clara   Infor Databases 1
                    OCaml     2
5555  Käthe   Art   Apel      1
                    Repin     1
6666  Claudia Sport Psycho    2
                    Ski       1
```

Intermediate result after the first gib-clause:

```
1111  Ernst   Math  Algebra   1
1111  Ernst   Math  History   1
1111  Ernst   Math  Logic     2
2222  Sophia  Infor Algebra   3
2222  Sophia  Infor Databases 1
2222  Sophia  Infor Otto      1
3333  Clara   Infor Databases 1
3333  Clara   Infor OCaml     2
5555  Käthe   Art   Apel      1
5555  Käthe   Art   Repin     1
6666  Claudia Sport Psycho    2
6666  Claudia Sport Ski       1
```

To get the intermediate result, STID segments are tried to be inserted first. This is not possible because there is no exams-data on this level. Then again, each COURSE segment is extended by its first level parent data. This data is inserted exam by exam and student by student.

Program 10.3.9: For each name, output the "first" MARK entry or "null value" if no MARK entry is present. Print the other collections for comparison purposes.

```
aus students.tab
gib (NAME,MARK? m),(NAME,MARK m),(NAME,MARK b),(NAME,MARKb m)
```

Result (tabh)

```
NAME,   MARK? m, (NAME,   MARK m), (NAME,   MARK b), (NAME,   MARKb m)
Clara   1        Clara    1        Clara    1        Clara    1 2
Claudia 2        Clara    2        Clara    2        Claudia  1 2
Ernst   1        Claudia  1        Claudia  1        Ernst    1 1 2
Käthe   1        Claudia  2        Claudia  2        Käthe    1 1
Sophia  3        Ernst    1        Ernst    1        Sophia   1 1 3
Ulrike           Ernst    2        Ernst    1        Ulrike
                 Käthe    1        Ernst    2
                 Sophia   1        Käthe    1
                 Sophia   3        Käthe    1
                                   Sophia   1
                                   Sophia   1
                                   Sophia   3
```

STID segments can be inserted in the first and last collection, since only names are required.

Program 10.3.10: (Restructuring) Reverse the given structuring. I.e., swap COURSE and NAME.

```
aus students.tab
gib COURSE,(NAME,MARK b)m
```

Result (tab)

```
COURSE , (NAME ,  MARK b) m
Algebra   Ernst   1
          Sophia  3
Apel      Käthe   1
Databases Clara   1
          Sophia  1
```

```
History   Ernst   1
Logic     Ernst   2
OCaml     Clara   2
Otto      Sophia  1
Psycho    Claudia 2
Repin     Käthe   1
Ski       Claudia 1
```

Here, an attempt is first made to insert the STID segments. Since no COURSE attribute exists, they cannot be inserted. Therefore, the extended COURSE segments are inserted first at the COURSE level and then at the NAME level.

Program 10.3.11: (Restructuring with additional tags) Reverse the given structuring by changing COURSE from inner to outer collection and NAME from outer to inner. Create additional tuple and sub-tuple tags.

```
aus students.tab
gib COURSES
    COURSES     = COURSETUPLEm
    COURSETUPLE = COURSE,EXAMSTUPLEb
    EXAMSTUPLE  = NAME,MARK
```

Result (ment)

```
<COURSES>
  <COURSETUPLE>
    <COURSE>Algebra</COURSE>
    <EXAMSTUPLE>
      <NAME>Ernst</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
    <EXAMSTUPLE>
      <NAME>Sophia</NAME>
      <MARK>3</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Apel</COURSE>
    <EXAMSTUPLE>
      <NAME>Käthe</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Databases</COURSE>
    <EXAMSTUPLE>
      <NAME>Clara</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
    <EXAMSTUPLE>
      <NAME>Sophia</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>History</COURSE>
    <EXAMSTUPLE>
      <NAME>Ernst</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Logic</COURSE>
    <EXAMSTUPLE>
      <NAME>Ernst</NAME>
      <MARK>2</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>OCaml</COURSE>
    <EXAMSTUPLE>
      <NAME>Clara</NAME>
      <MARK>2</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Otto</COURSE>
    <EXAMSTUPLE>
      <NAME>Sophia</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Psycho</COURSE>
    <EXAMSTUPLE>
      <NAME>Claudia</NAME>
      <MARK>2</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Repin</COURSE>
    <EXAMSTUPLE>
      <NAME>Käthe</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
  <COURSETUPLE>
    <COURSE>Ski</COURSE>
    <EXAMSTUPLE>
      <NAME>Claudia</NAME>
      <MARK>1</MARK>
    </EXAMSTUPLE>
  </COURSETUPLE>
</COURSES>
```

Now, we want to illustrate the set-theoretic operations union, intersection, and set-difference. Since the STID column in students.tab is already a union, we illustrate the union with the files exams1 and projects1.

Program 10.3.12: Construct the union of two files, where each value of each file should appear in the result.

```
aus exams1.tab,projects1.tab    # a pair of tables
```

```
gib STIDb
```

Result (tabh)

```
STIDb
1111 1111 1111 1111 1111 2222 2222 2222 2222 2222 2222 3333 3333 4444
5555 5555 5555 6666 6666 6666 6666
```

If we replace b with m in the gib statement, duplicates are eliminated.
Result

```
STIDm
1111 2222 3333 4444 5555 6666
```

If we want to know from which file each STID comes from, we can add corresponding information

```
aus exams1.tab,projects1.tab       # a pair of tables
gib STID,COURSE?,PROJ? b
```

Result (tab)

```
STID,COURSE?,  PROJ? b
1111           Fritz
1111           Otto
1111 Algebra
1111 History
1111 Logic
2222           Ghandi
2222           Ming
2222           Otto
2222 Algebra
2222 Databases
2222 Otto
3333 Databases
3333 OCaml
4444            Monet
5555 Monet
5555 Apel
5555 Repin
6666           Matthes
6666           Witt
6666 Psycho
6666 Ski
```

Program 10.3.13: Construct the intersection of two files with different schemas.

```
aus exams1.tab,projects1.tab
gib STID,COURSE?,PROJ? m
gib STID,COURSE,PROJ m
# This restructuring can also be realized through selections
gib STIDm
```

Result (tabh)

```
STIDm
1111 2222 5555 6666
```

Intermediate result after the first gib statement

```
STID, COURSE?, PROJ? m
1111 Algebra   Fritz
2222 Algebra   Ghandi
3333 Databases
4444           Monet
5555 Apel      Monet
6666 Psycho    Matthes
```

Program 10.3.14: Set difference: Specify all STIDs of exams1.tab that are not contained in projects1.tab.

```
aus exams1.tab
rename STID ! STUDID
,projects1.tab # in turn results in a tuple (pair) of tables
sel- STUDID in STIDm
gib  STUDIDm
```

Result (tab)

```
STUDIDm
3333
```

Program 10.3.15: query 10.3.14, but with nested query

```
aus  exams1.tab
sel- STID in begin projects1.tab;; gib STIDm end
gib  STIDm
```

Result (tab)

```
STUDIDm
3333
```

Program 10.3.16: (Grouping with Aggregation) Calculate the number of students and the number for each faculty. Sort the students by FAC and NAME.

```
aus students.tab
gib CNT,(FAC,CNT,(NAME,STID b)m)
    CNT:= STID! ++1
```

Result (tab)

```
CNT,  FAC , CNT,  (NAME ,  STID b) m
6     Art   2      Käthe   5555
                   Ulrike  4444
      Infor 2      Clara   3333
                   Sophia  2222
      Math  1      Ernst   1111
      Sport 1      Claudia 6666
```

Program 10.3.17: (Restructuring with Aggregation) Give the total of all scholarships and the total for each course. Sort the records by course.

```
aus students.tab
gib SU,(COURSE,SU m)
    SU:= STIP ! ++
```

Result (tab)

```
SU ,(COURSE ,  SU m)
2950 Algebra   900
     Apel      600
     Databases 850
     History   500
     Logic     500
     OCaml     450
     Otto      400
     Psycho    600
     Repin     600
     Ski       600
```

It is interesting to note here that the "sum" of the inner SU values is generally larger than the outer SU value. This is due to the fact that a particular course usually occurs in more than one student record.

Program 10.3.18: Search the name of the student with ID 2222.

```
aus  students.tab
sel  STID = 2222
gib  NAME
```

Result (tab)

```
NAME
Sophia
```

Program 10.3.19: Divide the students of the all faculties except sport into independent tables.

```
aus students.tab
sel- Sport
gib FAC,(NAME,LOC? m)m
cut 1   # the second argument says that only one faculty is desired
        # for each sub table
```

Result (tab)

```
FAC ,(NAME  ,LOC?  m) l,(FAC ,(NAME  ,LOC?  m) l),(FAC ,(NAME ,LOC? m) l)
Art    Käthe  Gerwisch     Infor Clara  Oehna         Math   Ernst Oehna
       Ulrike                    Sophia Berlin
```

The concept of hierarchical path is important for all operations. Its definition is based on "narrow" schemes. All collection symbols except '?' are real collection symbols.
A **schema s is narrow** if for any 2 real collections c and c' holds, either c is contained in c' or c' is contained in c. Fields f1 and f2 of a schema s are on a **hierarchical path** (**HP for** short) **with respect to s** if the schema formed by forgetting all fields except f1and f2 is narrow.
X,Ym,Zm m is not narrow, but X,Y?,Z? m is. PROJ and COURSE are in
NAME, (COURSE,MARK m),(PROJ,HOURS m)m, not on a hierarchical path; unlike PROJm and COURSE. This is visible in the graphical representation of the schema.

```
                     m
                     |
       (NAME, m,                m)
             |                  |
       (COURSE,MARK)       (PROJ,HOURS)
```

Program 10.3.20: Put simply two fields that are not on one HP onto one HP

```
aus students.tab
gib COURSE,PROJ m # is empty
```

Result (tabh)

```
COURSE, PROJ m
```

Program 10.3.21: Sort and group the students, who have taken an Algebra COURSE by their corresponding marks and sort them by name and print all their projects.

```
aus students.tab
sel COURSE=Algebra
gib MARK,(NAME,(PROJ,HOURS m)b) m
```

Result (tab)

```
MARK, (NAME, (PROJ,  HOURS m) b) m
1      Ernst  Fritz  4
              Otto   2
3      Sophia Ghandi 5
              Ming   4
              Otto   6
```

Although PROJ and MARK are not on one HP, the project collection is not empty. This is possible, because the operation *strip* is applied, if the target scheme contains fields, which are not on a hierarchical path. Strip generates a table of type:
STID ,NAME ,LOC ,STIP ,FAC ,COURSE ,MARK ,(PROJ ,HOURS m) m
Here, MARK and COURSE are on a HP.

10.4 Joining by nested queries

The horizontal merging or joining of the information of two tables is called a "join". In our approach, joining two flat tables is not necessarily a flat table. From point of view of power we do not need an additional join operation. Meaningful structures can be created with assignments (:=).

Program 10.4.1: “Add” exams1 data to students1 data.

```
aus students1.tab
:= exams1.tab at FAC
```

Initial part of the result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(STID, COURSE ,  MARK m) m
1111  Ernst   Oehna    500   Math  1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
                                   2222  Algebra   3
                                   2222  Databases 1
                                   2222  Otto      1
                                   3333  Databases 1
                                   3333  OCaml     2
                                   5555  Apel      1
                                   5555  Repin     1
                                   6666  Psycho    2
                                   6666  Ski       1
2222  Sophia  Berlin   400   Infor 1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
                                   2222  Algebra   3
                                   2222  Databases 1
                                   2222  Otto      1
                                   3333  Databases 1
                                   3333  OCaml     2
                                   5555  Apel      1
                                   5555  Repin     1
                                   6666  Psycho    2
                                   6666  Ski       1
3333  Clara   Oehna    450   Infor 1111  Algebra   1
                                   1111  History   1
                                   1111  Logic     2
                                   2222  Algebra   3
                                   ...
...
```

The result contains 6*12=72 sub tuples. To get the 12 desired sub tuples, we need to add a condition:

Program 10.4.2: Add all corresponding exams1 records to each students1 record ("Structured left outer join").

```
aus students1.tab
:=exams1.tab at FAC
sel  COURSE! STUDENTS1/STID=EXAMS1/STID
```

Result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(STID, COURSE ,  MARK m) m
1111  Ernst   Oehna    500   Math  1111  Algebra    1
                                   1111  History    1
                                   1111  Logic      2
2222  Sophia  Berlin   400   Infor 2222  Algebra    3
                                   2222  Databases  1
                                   2222  Otto       1
3333  Clara   Oehna    450   Infor 3333  Databases  1
                                   3333  OCaml      2
4444  Ulrike           400   Art
5555  Käthe   Gerwisch 600   Art   5555  Apel       1
                                   5555  Repin      1
6666  Claudia Berlin   600   Sport 6666  Psycho     2
                                   6666  Ski        1
```

If we want to omit Ulrike, we just have to omit the level identifier 'COURSE:'. Each tabment with the name xyz.tab has the outermost tag XYZ. Therefore, the above extension results in the following TT:

TT (tabment type of the result)

```
TABMENT! STUDENTS1
STUDENTS1! STID,NAME,LOC?,STIP,FAC,EXAMS1 m
EXAMS1! STID,COURSE,MARK m
MARK STIP STID! ZAHL
COURSE FAC LOC NAME! TEXT
```

This TT allows accurate specification of column names despite duplicate name occurrences. EXAMS1/COURSE is the same as COURSE, because COURSE appears only once on the right side of the TT.

In addition, STUDENTS1 is on the right side of EXAMS, so the tag path STUDENTS1/EXAMS/COURSE is also identical to COURSE. However, the "tag path" STID does not specify exactly, since it occurs twice. STUDENTS1/STID is the student identifier of the students1 table and EXAMS1/STID=STUDENTS1/EXAMS1/STID of the exam table.

In an access path X/Y/Z, Z must occur on the right side of Y and Y must occur on the right side of X in the TT. X is the paternal marker of Y and Y is the paternal marker of Z. There is no tag between X and Y and Y and Z (in the xml- or ment-representation). If we don't know all the intermediate tags, we can also use the X//Z notation. In this case, there can be any number of tags between X and Z. That is, this tag path corresponds to a complete tag path X/X1/X2/.../Xn/Z for matching tags X1,...,Xn. Therefore STUDENTS1//COURSE describes COURSE in the same way as the full tag path STUDENTS1/EXAMS1/COURSE.

Program 10.4.3: Program with nested query

```
aus students1.tab
EXAMS:=begin aus exams1.tab;;sel  STID=STID~ end at FAC
```

Result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(STID, COURSE ,  MARK m) m
1111  Ernst   Oehna    500   Math  1111  Algebra    1
                                   1111  History    1
                                   1111  Logic      2
2222  Sophia  Berlin   400   Infor 2222  Algebra    3
                                   2222  Databases  1
                                   2222  Otto       1
3333  Clara   Oehna    450   Infor 3333  Databases  1
                                   3333  OCaml      2
4444  Ulrike           400   Art
5555  Käthe   Gerwisch 600   Art   5555  Apel       1
                                   5555  Repin      1
6666  Claudia Berlin   600   Sport 6666  Psycho     2
                                   6666  Ski        1
```

Nested queries are contained in *begin* and *end*. If we want to refer to a column name outside the inner query, we must add a "~". Therefore STID~ is the identifier of STID of students1.

Program 10.4.4: Attempts to generate the given student table from three given flat relations..

```
aus students1.tab
PR:=begin aus projects1.tab
          sel STID=STID~
          gib PROJ,HOURS m end at FAC
EX:=begin aus exams1.tab
          sel STID=STID~
          gib COURSE,MARK m end at FAC
```

Result (tab)

```
STID, NAME ,  LOC?    ,STIP, FAC ,(COURSE ,  MARK m ),(PROJ ,   HOURS m) m
1111  Ernst   Oehna    500   Math  Algebra    1         Fritz     4
                                   History    1         Otto      2
                                   Logic      2
2222  Sophia  Berlin   400   Infor Algebra    3         Ghandi    5
                                   Databases  1         Ming      4
                                   Otto       1         Otto      6
3333  Clara   Oehna    450   Infor Databases  1
                                   OCaml      2
4444  Ulrike           400   Art                        Monet    10
5555  Käthe   Gerwisch 600   Art   Apel       1         Monet    20
                                   Repin      1
6666  Claudia Berlin   600   Sport Psycho     2         Matthes   8
                                   Ski        1         Witt     12
```

The result corresponds to students.tab.

Program 10.4.5: Generate a table with three nested levels.

```
aus   facs.tab
proj- STUDCAPACITY
ST:=begin aus students1.tab
    sel   FAC=FAC~
    proj- FAC end at BUDGET
EX:=begin aus exams1.tab
    sel   STID=STID~
    proj- STID end at STIP
```

Result (tab)

```
FAC  ,DEAN , BUDGET,  (STID, NAME , (LOC?    ,STIP,((COURSE ,  MARK m) m) m) m
Art   Sitte    2000    4444  Ulrike          400
                       5555  Käthe   Gerwisch 600    Apel      1
                                                     Repin     1
Infor Reichel 10000    2222  Sophia  Berlin   400    Algebra   3
                                                     Databases 1
                                                     Otto      1
                       3333  Clara   Oehna    450    Databases 1
                                                     OCaml     2
Math  Dassow   1000    1111  Ernst   Oehna    500    Algebra   1
                                                     History   1
                                                     Logic     2
Philo Hegel    1000
Sport Streich  8000    6666  Claudia Berlin   600    Psycho    2
                                                     Ski       1
```

If we want to delete only a few columns, we can use the *proj-* (projection) clause instead of a *gib* clause. We notice that we get a structure with nesting depth 3, although the deepest nesting level in the program is 2.

10.5 A user-friendly “join” (ext ext2)

Through Example 10.3.20, it has become clear that the problem of loading data onto an HP when it is not yet on an HP in the source structure can be solved in some situations with an additional *gib* statement without using the Cartesian product. This problem is even more important when we consider a given relational database with flat structures. In a tuple of such files, nothing is on an HP except the fields that are in the same table. Therefore, an ordinary *gib* statement is not very expressive. Relational systems solve this problem with *joins*. But the *join* is related to the Cartesian product. Moreover, join-conditions have to be used. In [Gol08] experiments with students were described. They showed that missing join conditions are the most common semantic SQL error. If we use both constructs of this section, the join conditions generally do not need to be written. In the first part of this section, we present some typical queries for *ext* (extension). It is easy to use *ext*, but its definition seems to be a bit more complicated than its application.

Program 10.5.1: Give the very good exams and the time-intensive projects for all students who do not live in Gerwisch, who completed a COURSE with a 1, and who have the time-intensive projects. Group the students by place of residence.

```
aus  students1.tab ext exams1.tab ext projects1.tab
sel- LOC=Gerwisch
sel  MARK=1
sel  HOURS>2
gib  LOC,(NAME,(COURSE,MARK m),(PROJ,HOURS m)b)m
```

Result (tab)

```
LOC , (NAME , (COURSE ,  MARK l), (PROJ ,  HOURS m) b) m
Berlin Claudia Ski       1         Matthes  8
                                   Witt    12
       Sophia  Databases 1         Ghandi   5
               Otto      1         Ming     4
                                   Otto     6
Oehna  Ernst   Algebra   1         Fritz    4
               History   1
```

Program 10.5.2: Group and sort student names with bad grades by faculty and output the students' bad courses. Omit STUDCAPACITY.

```
aus facs.tab,students1.tab,exams1.tab ext2
sel MARK>2
proj- STUDCAPACITY
```

Result (tab)

```
FAC  ,DEAN   ,BUDGET ,(STID ,NAME  ,LOC?  ,STIP ,(COURSE ,MARK m) m) m
Infor Reichel 10000    2222  Sophia Berlin 400    Algebra 3
```

Program 10.5.3: Give out all students from Oehna with dean, courses and projects..

```
aus facs.tab ext students.tab
sel LOC=Oehna
gib STID,NAME,FAC,DEAN,COURSEm,PROJm m
```

Result (tab)

```
STID, NAME, FAC,  DEAN,   COURSEm,  PROJm  m
1111  Ernst Math  Dassow  Algebra   Fritz
                          History   Otto
                          Logic
3333  Clara Infor Reichel Databases
                          OCaml
```

Program 10.5.4: Add the teacher column to the courses of the students of the computer science faculty.

```
aus students.tab ext courses.tab
sel FAC=Infor
gib NAME,LOC?,(COURSE,TEACHER,MARK m),PROJm m
```

Result (tab)

```
NAME  , LOC?, (COURSE  ,  TEACHER,MARK m), PROJm m
Clara  Oehna  Databases Saake   1
Sophia Berlin Algebra   Reichel 3          Ghandi
              Databases Saake   1          Ming
              Otto      Benecke 1          Otto
```

Program 10.5.5: Find all students from large faculties who have a good mark in algebra. FAC ,(LOC ,NAMEb m) m . Structure students by FAC and LOC, and sort them by NAME.

```
aus facs.tab,students1.tab,exams1.tab ext2
sel STUDCAPACITY>300
sel MARK<4 & COURSE=Algebra
gib FAC,(LOC,NAMEb m)m
```

Result (tab)

```
FAC ,(LOC   ,NAMEl m) m
Infor Berlin Sophia
```

11 Special Restructuring Operations

11.1 The Bill of Material Problem (BOM) (onrs)

The onrs operation was introduced to provide o++o numbers for solving BOM problems. The given tabment must be of type
X1,...,Xn,(Y1, ... Yk m) m
where X1 and Y1 are the keys of the respective collections. BOM-problems occur often in industry. Surely, not only very large data sets have to be handled. Below, it can be seen that the whole BOM is stored in one structured table. Both collections of the input-table are sets. That means we have direct access to each tuple and sub-tuple, if the part or part-number is given. In the first step otto-numbers are generated. We shall see that these numbers are also important for structured texts like books or the Wikipedia. The operation *nextonr* is similar to *next*, but it ends already, if an *ottonr* of the same or smaller length follows. The rest is realized by *gib*.

Program 11.1.1: Print the BOM of the car Wartburg.

```
<TAB!
PART,    PROPERTY,  (SUBPART,   COUNT m) m
Bushing  cylindrical
Engine   heavy        Piston     6
                      Screw      8
Piston   light        Bushing    1
                      PistonRing 2
Rim      smooth
Trabant  modern       Body       1
                      Engine     1
                      Wheel      4
Wartburg fast         Body       1
                      Climate    1
                      Engine     1
                      Wheel      4
Wheel    round        Rim        1
                      Screw      5
                      Tire       1
!TAB>
onrs Wartburg
COUNTOTTO:= COUNT nextonr
            COUNTOTTO pred *COUNT at COUNT
gib SUBPART,TOTAL m TOTAL:= COUNTOTTO!++
```

Result (tab)

```
SUBPART,   TOTAL m
Body         1
Bushing      6
Climate      1
Engine       1
Piston       6
PistonRing  12
Rim          4
Screw       28
Tire         4
Wheel        4
```

Intermediate result of line “onrs Wartburg” (tab)

```
PART    ,PROPERTY ,(OTTONR ,SUBPART   ,COUNT  m) l
Wartburg fast       1       Body        1
                    2       Climate     1
                    3       Engine     1
                    3.1     Piston     6
                    3.1.1   Bushing    1
                    3.1.2   PistonRing 2
                    3.2     Screw      8
                    4       Wheel      4
                    4.1     Rim        1
                    4.2     Screw      5
                    4.3     Tire       1
```

Intermediate result without last line (tab)

```
PART    ,PROPERTY ,(OTTONR ,SUBPART    ,COUNT ,COUNTOTTO  m) l
Wartburg fast       1       Body       1       1
                    2       Climate    1       1
                    3       Engine     1       1
                    3.1     Piston     6       6
                    3.1.1   Bushing    1       6
                    3.1.2   PistonRing 2      12
                    3.2     Screw      8       8
                    4       Wheel      4       4
                    4.1     Rim        1       4
                    4.2     Screw      5      20
                    4.3     Tire       1       4
```

It can be seen that all direct subparts from the Wartburg are assigned an otto number, which consists of only one number. The engine is one such part. The direct lower parts of the engine (screw and piston) are assigned otto numbers with two digits. Similarly, the direct lower parts of the piston are assigned otto numbers with 3 digits. Thus, a non-recursive set without redundancy is formed for the Golf. The table recursion could now be applied to this set to calculate the multiplicity of containing a subpart.
Beside the input tabment *onrs* needs one or more parts (here only Wartburg) for which the ONR resolution is to be made. Accordingly, the program line
onrs [Wartburg Trabant]
is correct and reasonable.

11.2 Transposing Matrices and structured Tables

Transposing data is well known from matrices, but it seems to be useful also for structured tables. For example, if a computer screen or a sheet of paper is not wide enough or too small, then often appropriate transpositions can help.

Program 11.2.1: Transpose a simple matrix with column names.

```
<TAB!
X1,X2 l
1  2
3  4
5  6
!TAB>
transpose
=:Y1,..,Y3 l
```

Final result:

```
Y1 ,Y2 ,Y3  l
1   3   5
2   4   6
```

Intermediate result of the above subprogram without the last program line

(OCaml-Term)

```
Coll_t(List,Tuple_s [Mixe_s;Mixe_s;Mixe_s],
 [Tuple_t [
      Tag0 ("X1",
         El_tab (Int_v (1))
      );
      Tag0 ("X1",
         El_tab (Int_v (3))
      );
      Tag0 ("X1",
         El_tab (Int_v (5))
      )
   ];
   Tuple_t [
      Tag0 ("X2",
         El_tab (Int_v (2))
      );
      Tag0 ("X2",
         El_tab (Int_v (4))
      );
      Tag0 ("X2",
         El_tab (Int_v (6))
      )
   ]
])
```

It is also possible to apply at first the operation *untagall* to the given table and then *transpose*. The result will be the same.

Now, we consider a table of marks, where the list of marks appears already in a transposed way, because the list of marks is arranged horizontally, but from logical point are they vertically arranged. To save further space a transposition of the (SUBJECT,MARKl m) – collections could be useful.

```
NAME  ,(SUBJECT ,MARK1 m) m
Clara  Chinese  2 4 3 5 3
       Maths    1 2 1 3
       Physics  1 1
Ernst  Chinese  1 2 1 4
       Latin    1 2 6
       Maths    2 4 3
       Physics  3 1 1
Sophia Chinese  1 2 3 1 2
       Maths    1 2 4 2
       Physics  3 1 4
```

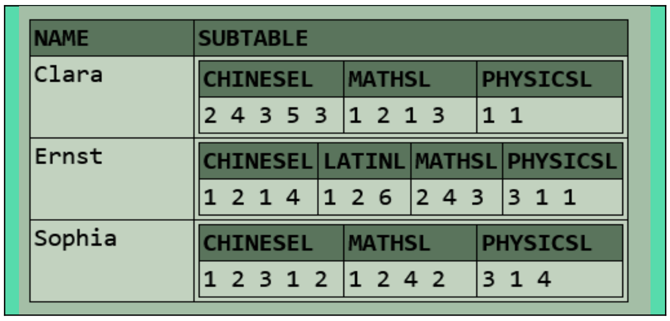
marks.tabh

Program 11.2.2: Arrange each inner set horizontally.

```
marks.tabh
tag SUBTABLE ! (SUBJECT,MARK1 m)
SUBTABLE::= SUBTABLE transpose meta1
```

Result:

```
MARKS
```



NAME	SUBTABLE			
Clara	CHINESEL	MATHSL	PHYSICSL	
	2 4 3 5 3	1 2 1 3	1 1	
Ernst	CHINESEL	LATINL	MATHSL	PHYSICSL
	1 2 1 4	1 2 6	2 4 3	3 1 1
Sophia	CHINESEL	MATHSL	PHYSICSL	
	1 2 3 1 2	1 2 4 2	3 1 4	

The above result is a problem for o++o, because it is not an ordinary table. The second sub table has 4 components and the others only three. Therefore, for example, the tab-representations do not work. By the additional line

```
gib NAME,CHINESEl,MATHSl,PHYSICSl,LATINl m
```

the problem can be solved. In this case each student has an empty or non-empty LATIN-collection. *meta1* will be clear in the following example.

Program 11.2.3: Arrange each inner set horizontally without an additional tag.

```
marks.tabh
:= NAME tup nth 2 transpose meta1
gib NAME,CHINESEl,MATHSl,PHYSICSl,LATINl m
```

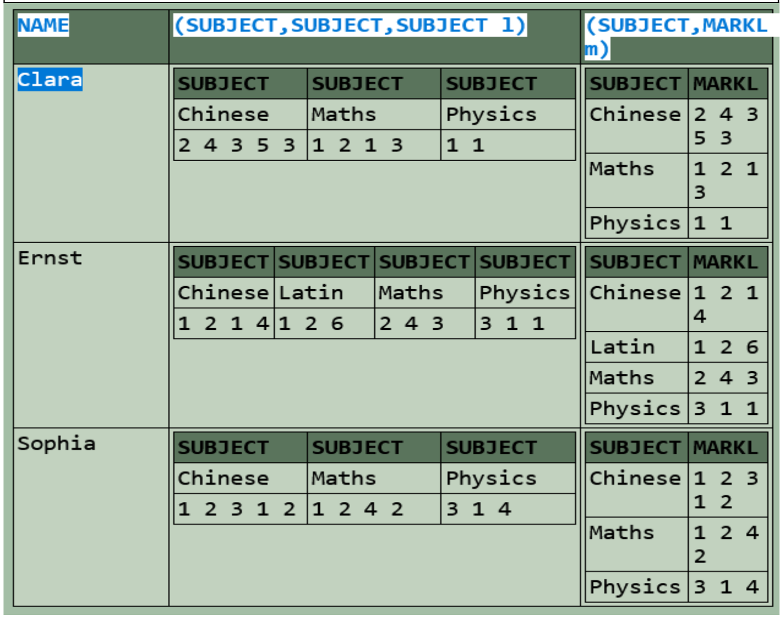
Final result:

```
NAME,  CHINESEl, MATHSl, PHYSICSl, LATINl m
Clara  2 4 3 5 3 1 2 1 3 1 1
Ernst  1 2 1 4   2 4 3   3 1 1     1 2 6
Sophia 1 2 3 1 2 1 2 4 2 3 1 4
```

Sub program

```
marks.tabh
:= NAME tup nth 2 transpose
```

Intermediate result of the above subprogram (webh)



NAME	(SUBJECT,SUBJECT,SUBJECT l)				(SUBJECT,MARKL m)	
Clara	SUBJECT	SUBJECT	SUBJECT		SUBJECT	MARKL
	Chinese	Maths	Physics		Chinese	2 4 3 5 3
	2 4 3 5 3	1 2 1 3	1 1		Maths	1 2 1 3
					Physics	1 1
Ernst	SUBJECT	SUBJECT	SUBJECT	SUBJECT	SUBJECT	MARKL
	Chinese	Latin	Maths	Physics	Chinese	1 2 1 4
	1 2 1 4	1 2 6	2 4 3	3 1 1	Latin	1 2 6
					Maths	2 4 3
					Physics	3 1 1
Sophia	SUBJECT	SUBJECT	SUBJECT		SUBJECT	MARKL
	Chinese	Maths	Physics		Chinese	1 2 3 1 2
	1 2 3 1 2	1 2 4 2	3 1 4		Maths	1 2 4 2
					Physics	3 1 4

By *meta1* the (primary) data of the first element are taken as metadata. The following *gib* statement omits the SUBJECT and MARK columns and introduces a column LATIN for Clara and Sophia. Without this additional column, the data cannot be represented, for example in tab or tabh-format.

Program 11.2.4: Arrange the result of the previous query vertically.

```
<TABH!
NAME,  CHINESEl, MATHSl, PHYSICSl, LATINl m
Clara  2 4 3 5 3 1 2 1 3 1 1
Ernst  1 2 1 4   2 4 3   3 1 1     1 2 6
Sophia 1 2 3 1 2 1 2 4 2 3 1 4
!TABH>
SUBJECT,MARKl l:=NAME tup nths (2 .. 5)
                 transpose metaprim
gib NAME,(SUBJECT,MARKl m)m
SUBJECT::=SUBJECT subtext 1!1 + (SUBJECT
    subtext 2!(SUBJECT ++1- 1) lowercase)
```

Final result (nearly equal to `marks.tabh`)

```
NAME ,(SUBJECT, MARKl  m) m
Clara  Chinese  2 4 3 5 3
       Latin
       Maths    1 2 1 3
       Physics  1 1
Ernst  Chinese  1 2 1 4
       Latin    1 2 6
       Maths    2 4 3
       Physics  3 1 1
Sophia Chinese  1 2 3 1 2
       Latin
       Maths    1 2 4 2
       Physics  3 1 4
```

Example for the operation metaprim

```
MARKl := 1 2 1 3
metaprim
```

result (tab)

```
WORT ,ZAHLl
MARK  1 2 1 3
```

metaprim is only defined for one column tables.

The **transpose** operation can be used especially to convert tuples or sub tuples into lists. This has the advantage that list operations, such as selection, can be applied to these lists, too. The following file was obtained from an EXCEL table.

```
ID                    ,LAND      ,WIDTH  ,LENGTH ,HEIGHT ,JAN ,FEB  ,MRZ  ,APR  ,MAY  ,JUN  ,JUL  ,AUG  ,SEP  ,OCT  ,NOV ,DEC  l
BG0001a-Varna          Bulgaria   43.21   27.91     44    63.   68.   81.   87.   88.   81.   86.  100.   95.   88.  66.  59.
BG0002a-Shumen         Bulgaria   43.283  26.933   242    59.   68.   83.   88.   88.   81.   88.   98.   97.   90.  64.  57.
BG0003a-Ruse           Bulgaria   43.856  25.971    48    72.   79.   99.   97.   94.   87.   94.  105.  106.  103.  64.  57.
BG0004b-Veliko Tarnovo Bulgaria   43.086  25.656   137    61.   68.   85.   90.   86.   81.   87.   93.   96.   91.  65.  57.
BG0005b-Burgas         Bulgaria   42.51   27.47     31    64.   68.   84.   89.   86.   82.   88.   99.   94.   85.  65.  55.
BG0006a-Plovdiv        Bulgaria   42.15   24.75    171    88.   75.   87.   92.   84.   78.   83.   93.   98.   91.  67.  71.
BG0007a-Sofia          Bulgaria   42.697  23.323   560    52.   66.   79.   75.   75.   75.   82.   89.   89.   84.  53.  40.
BG0008a-Haskovo        Bulgaria   41.933  25.567   183    78.   76.   87.   95.   86.   80.   84.   95.   96.   93.  69.  65.
BG0009a-Blagoevgrad    Bulgaria   42.014  23.095   373    61.   71.   87.   86.   84.   80.   87.   98.  100.   86.  57.  48.
BY0001a-Minsk          Belarus    53.9    27.5     280    35.6  54.4  94.2  81.9  95.8  95.8  94.2  92.8  80.6  50.6 23.3 19.7
DE0001a-Norderney      Germany    53.71    7.15     11    32.7  57.8  75.9  96.5 101.2  85.7  91.5  93.7  79.2  64.  40.3 20.1
DE0002a-Husum          Germany    54.48    9.06      3    32.7  57.8  75.9  96.5 101.2  85.7  91.5  93.7  79.2  64.  40.3 20.1
DE0003a-Hamburg        Germany    53.64    9.99     11    29.8  51.1  63.2  82.8  91.5  75.6  84.1  87.   71.3  60.3 38.2 20.1
DE0004a-Hannover       Germany    52.47    9.68     59    29.8  51.1  63.2  82.8  91.5  75.6  84.1  87.   71.3  60.3 38.2 20.1
DE0005a-Kiel           Germany    54.34   10.09      4    29.8  51.1  63.2  82.8  91.5  75.6  84.1  87.   71.3  60.3 38.2 20.1
DE0006a-Arkona         Germany    54.68   13.44     42    21.6  45.7  72.9  95.  106.4  90.7  99.  100.4  85.7  64.7 37.4 20.8
DE0007a-Warnemünde     Germany    54.18   12.08      4    21.6  45.7  72.9  95.  106.4  90.7  99.  100.4  85.7  64.7 37.4 20.8
DE0008a-Potsdam        Germany    52.38   13.06     81    30.5  53.8  70.7  82.8  90.8  77.8  89.3  94.5  79.2  67.  38.2 22.3
DE0009a-Schwerin       Germany    53.64   11.39     59    30.5  53.8  70.7  82.8  90.8  77.8  89.3  94.5  79.2  67.  38.2 22.3
DE0010a-Teterow        Germany    53.76   12.56     38    30.5  53.8  70.7  82.8  90.8  77.8  89.3  94.5  79.2  67.  38.2 22.3
DE0011a-Braunschweig   Germany    52.29   10.45     88    34.2  55.1  69.2  83.5  93.   79.9  87.8  90.8  74.9  64.7 41.  22.3
```

```
DE0012a-Dresden      Germany   51.02   13.78   119    34.2  55.1  69.2  83.5  93.   79.9  87.8  90.8  74.9  64.7 41.  22.3
DE0013a-Wittenberg   Germany   51.89   12.65   105    34.2  55.1  69.2  83.5  93.   79.9  87.8  90.8  74.9  64.7 41.  22.3
DE0014a-Erfurt       Germany   50.98   10.96   316    37.2  53.8  71.4  79.2  85.6  76.3  84.1  85.6  71.3  63.2 36.  23.8
DE0015a-Harzgerode   Germany   51.65   11.14   404    37.2  53.8  71.4  79.2  85.6  76.3  84.1  85.6  71.3  63.2 36.  23.8
```
Part of a table **`climate_radiation.hsq in tab-format`**

climate_radiation has the scheme:
```
ID,COUNTRY,WIDTH,LENGTH,HEIGHT?,JAN,FEB,MRZ,APR,MAY,JUN,JUL,AUG,
                                                   SEP,OCT,NOV,DEC  l
```
It contains 17 columns. You can reduce the number of columns to 7 in the following way:

This flat table is transformed into a structured one, in which the radiations are arranged vertically and the months are output in an additional column:

Program 11.2.5: Transpose the radiations vertically.
```
climate_radiation.hsq
MON,RADIATION l:= JAN seg transpose metaprim
gib ID,LAND,WIDTH,LENGTH,HEIGHT,(MON,RADIATION l) l
```
Result (tab)
```
ID                   ,LAND      ,WIDTH  ,LENGTH ,HEIGHT ,(MON ,RADIATION  l) l
BG0001a-Varna         Bulgaria   43.21   27.91     44     JAN   63.
                                                          FEB   68.
                                                          MRZ   81.
                                                          APR   87.
                                                          MAY   88.
                                                          JUN   81.
                                                          JUL   86.
                                                          AUG  100.
                                                          SEP   95.
                                                          OCT   88.
                                                          NOV   66.
                                                          DEC   59.
BG0002a-Shumen        Bulgaria   43.283  26.933   242     JAN   59.
                                                          FEB   68.
                                                          MRZ   83.
                                                          APR   88.
                                                          MAY   88.
                                                          JUN   81.
                                                          JUL   88.
                                                          AUG   98.
                                                          SEP   97.
                                                          OCT   90.
                                                          NOV   64.
                                                          DEC   57.
. . .
```

Now you can easily create statistics about RADIATION or easily select specific months, etc.

Now, we show a transposition within a more complex given tabment. Here the elementary tag whose values are to become column names must be specified as the second input value.

Program 11.2.6: Arrange the subjects horizontally.
```
<TABH!
NAME,   LOC,    BORN,    CLASS?,(HOBBY,      HOURS l),(SUBJECT,MARKl l)l
Clara   Oehna   12.6.11  4        Riding     5          Math     1 2
                                  Chess      1          German   3 1 1
Claudia Dallgow 14.9.17           Chinese    5
```

```
                                  Food         4
Sophia  Dallgow 7.9.13   2        Painting     5         Math     1 2 1 1
                                  Wheelturning 4         German   1 2 1
                                  Chinese      6
!TABH>
tag X!(SUBJECT,MARKl l)
X::=X transpose meta1
gib NAME,LOC,BORN,CLASS?,(HOBBY,HOURS l),MATHl,GERMANl m
```

Result (tabh)

```
NAME,   LOC?,   BORN?,  CLASS?, (HOBBY,       HOURS  l),MATHl,  GERMANl l
Clara   Oehna   12.6.11 4        Riding       5         1 2     3 1 1
                                 Chess        1
Claudia Dallgow 14.9.17          Chinese      5
                                 Food         4
Sophia  Dallgow 7.9.13  2        Painting     5         1 2 1 1 1 2 1
                                 Wheelturning 4
                                 Chinese      6
```

12 Some operations for text processing with o++o (+ -+ cut satzl)

The + symbol can also be used to concatenate and manipulate text. Here, too, a small difference is made between TEXT and WORT:

Program 12.1: There are small differences between WORT and TEXT concatenation

```
WORDRESULT:=otto + " o++o"
TEXTRESULT:=otto text + " o++o"
```

Result (tab)

```
WORDRESULT, TEXTRESULT
otto_o++o   otto o++o
```

Since the first input value of WORDRESULT is a word, the result is also of type WORT. The same applies to the second case, where the result is a text. Words cannot contain spaces.

-+*text* is an operation with 3 input values. The TT of the first input value is retained or WORT is changed to TEXT, if a blank is inserted. Each occurrence of the second input value is replaced by the third.

Program 12.2: -+text example

```
<TAB!
X, Y 1
1 Today is Monday.
2 Yesterday is Sunday.
!TAB>
-+text "is S" ! "was S"
```

Result (tab)

```
X,Y 1
1 Today is Monday.
2 Yesterday was Sunday.
```

Program 12.3: text operations + - -+text

```
  TEXTPLUS:="Today is a beautiful " + day
  TEXTMINUS:=Thunmmder_weather - "m"
  TEXTMINUSPLUS:="Today is a beaoetiful day." -+text oe ! u
```

Result (tab)

```
TEXTPLUS,                 TEXTMINUS,      TEXTMINUSPLUS
Today is a beautiful day Thunder_weather Today is a beautiful day.
```

Program 12.4: "Coding" a text.

```
"Today is Tuesday. Tomorrow is Wednesday."
cut 1
sel- TEXT inmath ["a" "e" "i" "o" "u"]
TEXT::="t" if TEXT="m"!
       "m" if TEXT="t"!
       TEXT
++text
```

Result (tab)

```
TEXT
mdy s msdy. mtrrw s Wdnsdy.
```

Program 12.5: Eliminate the dot from numbers in a list.

```
Xl:=3 *l 5 ^ (1.1 ..6)
Y:= X cut 1
sel- WORT= "."
Y::=Y ++text
```

Result (tab)

```
X,             Y l
  3.3483695221 33483695221
 10.0451085663 100451085663
 30.1353256989 301353256989
 90.4059770967 904059770967
271.21793129   27121793129
```

A function for sentences (satzl) has been implemented, which so far uses a relatively rudimentary end-of-sentence detection.

Program 12.6: Disassemble a text into a list of sentences.

```
"Today is a beautiful day. Tomorrow, I will go to buy something."
satzl
```

Result (tab)

```
SATZl
Today is a beautiful day.
Tomorrow, I will go to buy something.
```

13 Format with o++o ('3 '4 norm3e norm3m mant rnd)

Analogous to SQL, o++o had initially limited itself to content problems. However, o++o uses much richer structures than SQL. Formatting was then taken over from SVG. Thus, one could frame a table, write letters bold or colored, etc. . Now we have implemented more possibilities. Numbers with a larger mantissa are hard to read if they are not grouped. Since many different variants are used for number representations in the world, we have chosen representations that do not collide with the existing ones as much as possible and are still better readable.

Grouping of digit sequences ('3 '4)

Following the Swiss model, o++o uses the apostrophe to make numbers better readable. Blocks of three are the most important along with blocks of four.

Program 13.1: Improve readability of several numbers by grouping them.

```
12345678, 1234567.87654 '3 ;1234567890 '4
```

```
Result (tab)
```

```
12'345'678 1'234'567.876'54 12'3456'7890
```

Such representations are created by the unary operations '3 and '4. The user can also set the apostrophe arbitrarily, for example to make telephone numbers more readable:

```
0176'84'208'408
```

Internationally, both the comma and the point (dot) are used as decimal separators and the point and the comma are also used for grouping. We hope to eliminate this inconsistency through these arrangements.

Exponent first notation (norm10m) and norm10e

PZAHL numbers with long mantissa are not to be grasped fast enough, since the more substantial exponent is indicated only at the end of the string. Furthermore, people think in thousands, millions, billions, ... An exponent 7 or 8 must be recognized as first 10- or 100-million. This way of thinking reflects o++o by allowing only multiples of 3 as exponent. Furthermore, the exponents can also be given first:

6m12.345 (12 million ...)

9m123.4 (123 billion ...)

the old mantissa first notation knows o++o nevertheless. However here also multiples of 3 are used as exponent:

12.3456789e6 (12 million ...)

123.456789e9 (123 billion ...)

These formatting's can be generated by the unary operations norm3m (for the representation with m) and norm3e (for the latter). The 3 expresses that the exponent is a multiple of 3.

Program 13.2: Improve the readability of numbers by normalizing the exponents.

```
X:=12345678.9 norm3m
Y:=12345678.9 norm3e
```

```
Result (tab)
X,          Y
6m12.3456789  12.3456789e6
```

The reduction of the digits (mant)

Most people don't care about the many decimal places when a calculator outputs the square root of 2 or 3 with more than 10 digits. The overload of irrelevant information makes it harder for us to see what is important. Therefore, omitting unnecessary digits (information) is important.

The binary function mant realizes this and converts the result immediately into the m-representation. I.e. the operation norm3m is applied at the same time. The second argument of mant specifies the number of digits desired.

Program 13.3: Reduce the number of digits to four.

```
12345678.98765 , 1234567890
mant 4
```

```
Result (web)
 6m12.34  9m1.234
```

14 Structured diagrams

With o++o you can easily create diagrams. Once you have created an o++o program, you can use the diagram button to open a new browser window that offers a choice of different diagram types. Column charts are certainly the most commonly used. The following rules apply to diagrams:

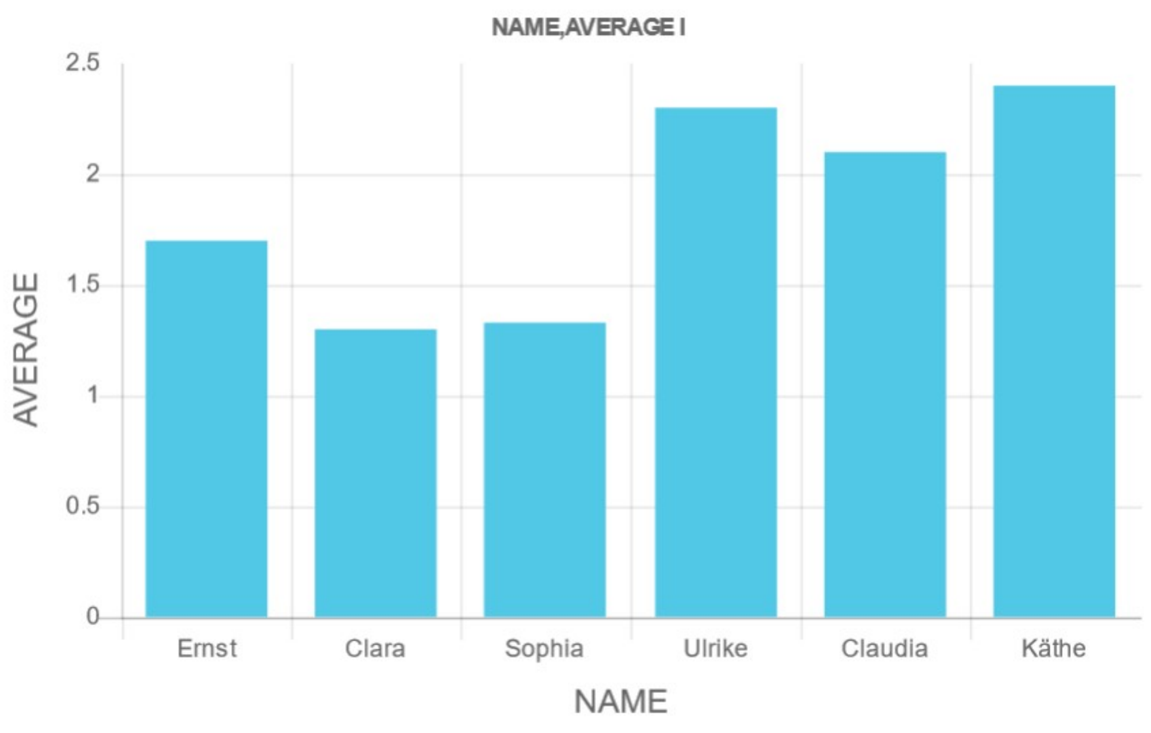
1. TEXTs are converted to words by the system by replacing each space with an underscore.
2. Numeric columns (ZAHL, PZAHL, RATIO) are displayed as columns.
3. The first word column of each hierarchy level is used as the signature for the columns. The other word columns of the level are ignored. If no word column exists, a dash acts as a signature.
4. If no RGB values are given, the system sets default colors. If the user wants to choose the colors, each numeric column must have an RGB column in the same level or higher. If an RGB value is placed directly in front of a number column, it determines the color of the column.
5. If the table to be displayed starts, with the column name TITEL, the content of the column is interpreted as the heading of the entire chart.

We already know that a simple list of numbers is an o++o program that can be represented as a diagram. If there is one more word in each row, it serves as a signature:

Program 14.1: Create a column chart with signatures

```
<TAB!
NAME,   AVERAGE l
Ernst   1.7
Clara   1.3
Sophia  1.33
Ulrike  2.3
Claudia 2.1
Käthe   2.4
!TAB>
```

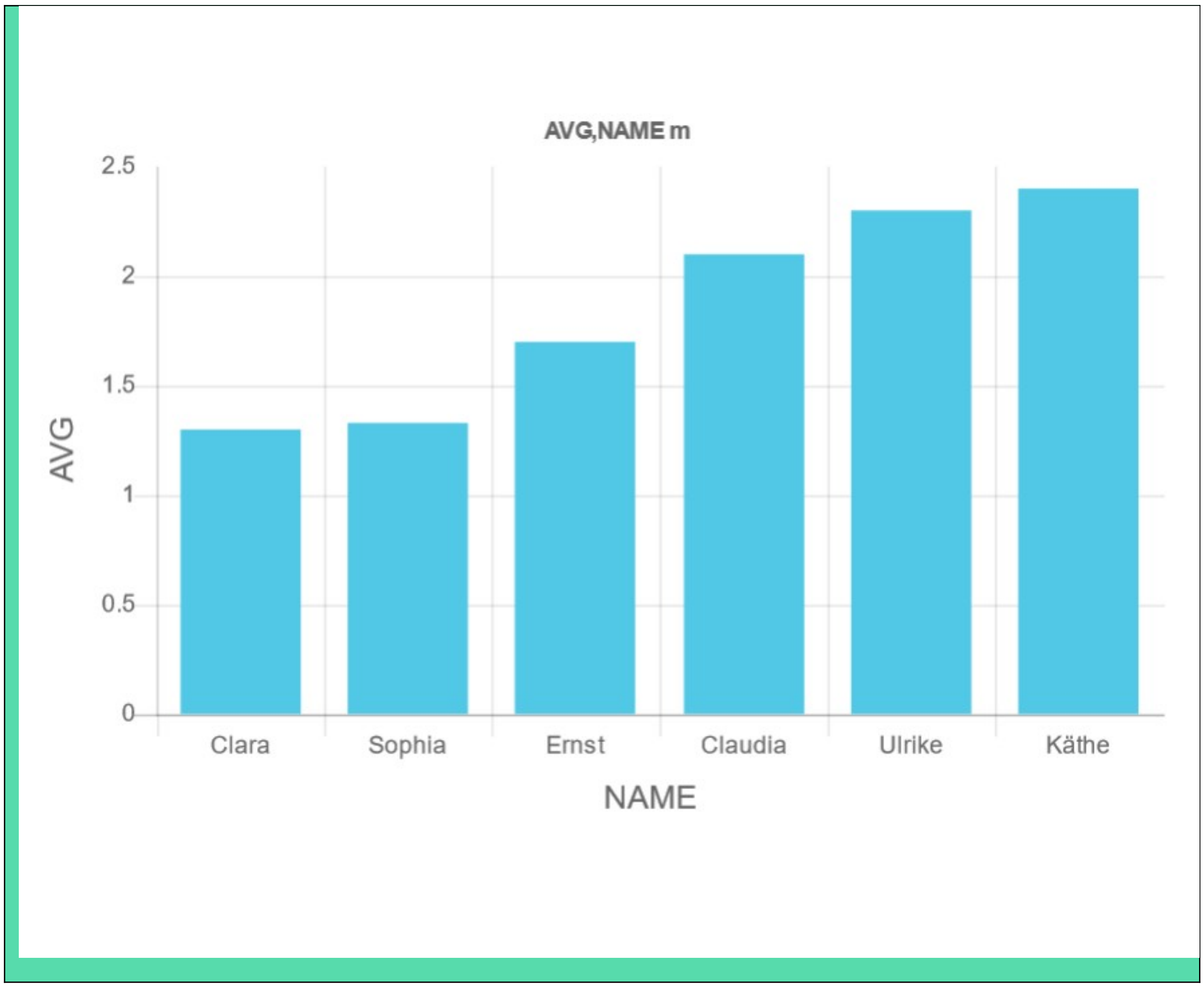
```
Result (struc.diagram- bar)
```



Program 14.2: Sort the columns with signatures by size

```
<TAB!
NAME,   AVG l
Ernst   1.7
Clara   1.3
Sophia  1.33
Ulrike  2.3
Claudia 2.1
Käthe   2.4
!TAB>
gib AVG,NAME m
```

```
Result (diagram - columns)
```



The following diagrams use the below table of towers.

TOWER,	CITY,	COUNTRY,	HEIGHT l
Burj Khalifa	Dubai	VAR	830
Shanghai Tower	Shanghai	China	632
Abraj Al Bait	Mecca	Saudi Arabia	601
Ping An Finance Center	Shenzen	China	599
Goldin Finance	Tainjin	China	597
Lotte World Tower	Seoul	South Korea	555
1 WTC	New York	USA	541
Guangzhou CTF Finance Center	Guangzhou	China	530
China Sun Tower	Beijing	China	528
Taipei 101	Taipei	Taiwan	508
World Finance Center	Shanghai	China	492
Lakhta Center	Saint Petersburg	Russia	462
Vincom Landmark 81	Ho Chi Minh City	Vietnam	461
Petronas Towers	Kuala Lumpur	Malaysia	452
Berlin TV Tower	Berlin	GDR	368

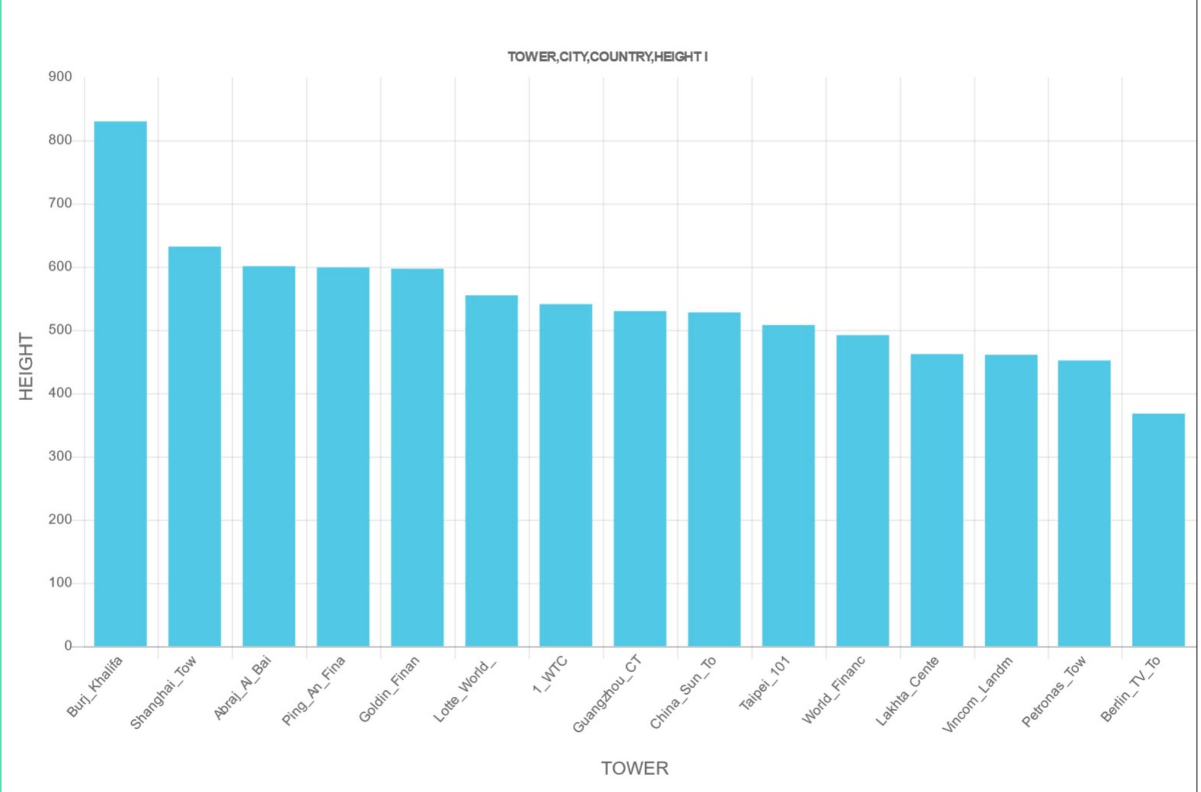
towers.tab

Program 14.3: Represent each tower by a column

```
towers.tab
TOWER::=TOWER subtext 1!12 # By this shortening of the name are also
                           # in the bar chart all names at the same time
```

```
                                     # visible; for space reasons, otherwise
                                     # sometimes some are hidden
```

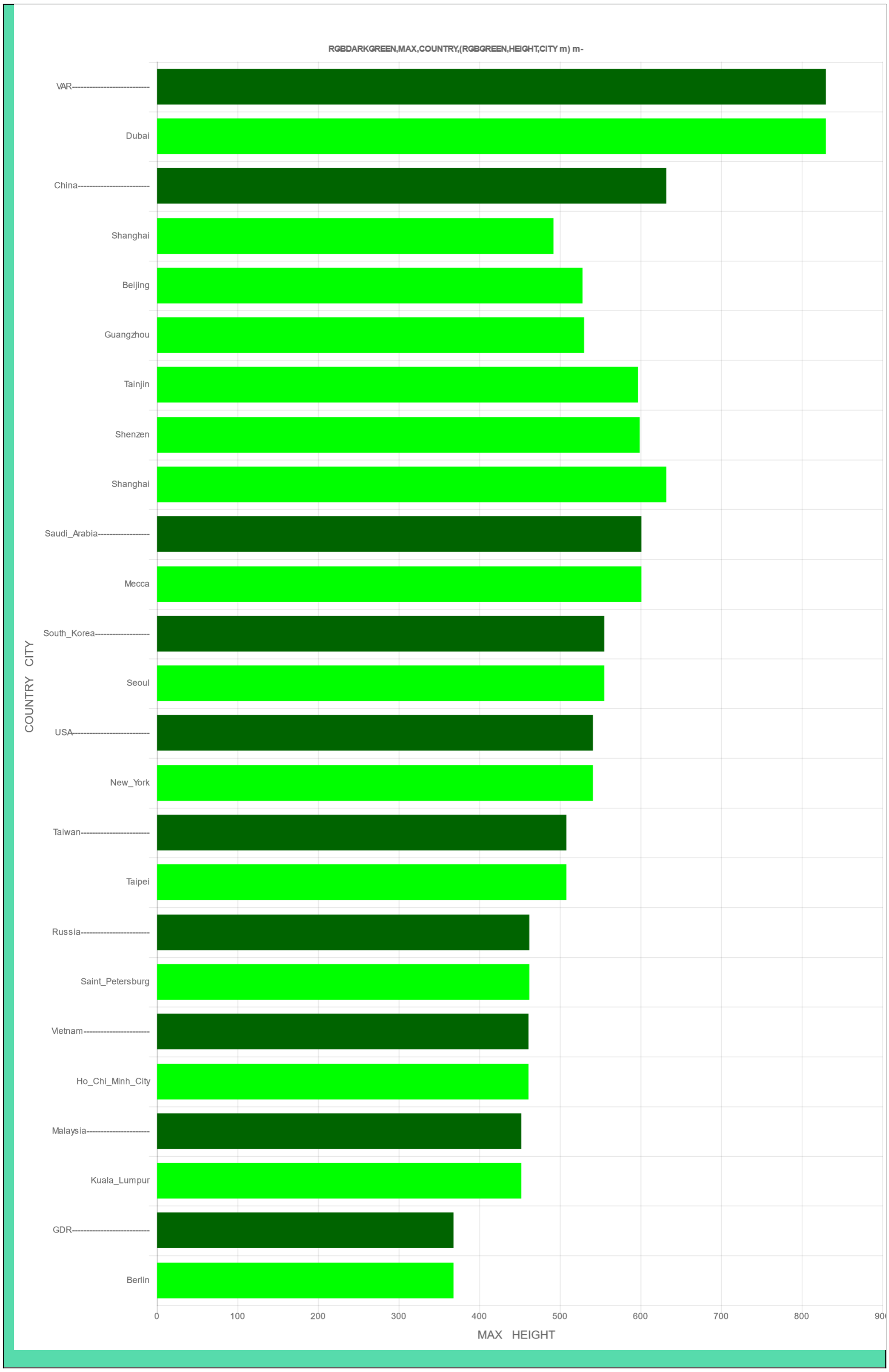
Result (diagram columns)



Program 14.4: Represent each tower by a bar and output the bars country by country. Countries with the highest towers are to be output first (sort downwards). For each country, sort the towers upwards. The countries are to be visually marked off.

```
aus towers.tab
gib MAX,COUNTRY,(HEIGHT,CITY m) m- MAX:=HEIGHT!max
COUNTRY::=COUNTRY wort + "-----------------------------------" subtext 1!
30
RGBDARKGREEN:=darkgreen leftat MAX
RGBGREEN:=green leftat HEIGHT
```

upper part of the result (struc.diagram bar)


RGBDARKGREEN,MAX,COUNTRY,(RGBGREEN,HEIGHT,CITY m) m-
VAR----------------------------
Dubai
China--------------------------
Shanghai
Beijing
Guangzhou
Tainjin
Shenzen
Shanghai
Saudi_Arabia-------------------
Mecca
South_Korea--------------------
Seoul
USA----------------------------
New_York
Taiwan-------------------------
Taipei
Russia-------------------------
Saint_Petersburg
Vietnam------------------------
Ho_Chi_Minh_City
Malaysia-----------------------
Kuala_Lumpur
GDR----------------------------
Berlin
COUNTRY CITY
0
100
200
300
400
500
600
700
800
900
MAX HEIGHT

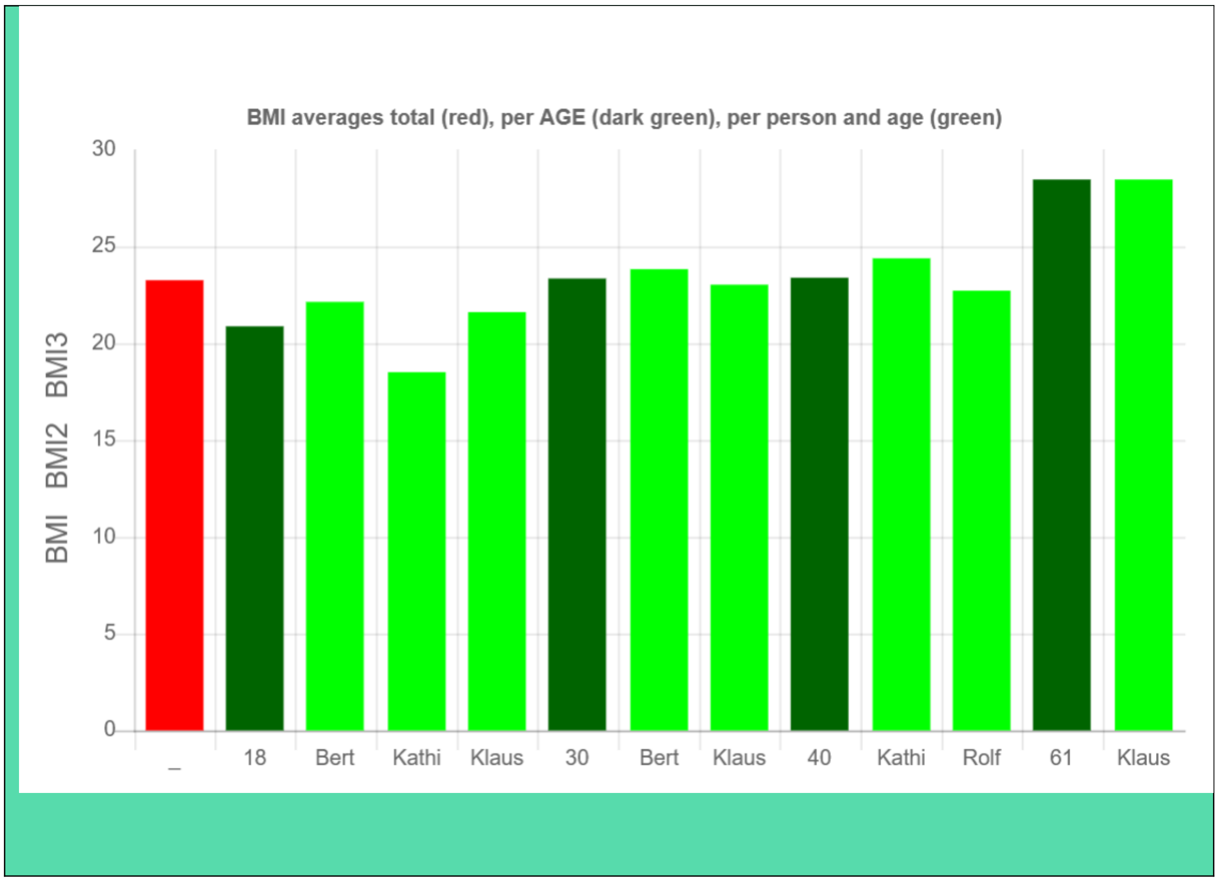
```
NAME,    LENGTH, (AGE,WEIGHTl l) l
Bert     1.72      18  66 65
                   30  70 71
Kathi    1.7       18  55 52
                   40  70 71
Klaus    1.68      18  61 60 62
                   30  65 63 67
                   61  80 82 79
Rolf     1.78      40  72 70 74
Victoria 1.61      13  51 50
Walleri  1.        3   16 15
```

weights.tabh

Program 14.5: Calculate BMI averages for all adults, for each age group, and overall. To realize the first 5 lines EXCEL needs more than 6 worksheets**.**

```
aus weights.tabh
sel NAME! AGE>20
gib BMI,(AGE,BMI,(NAME,BMI m) m)
    BMI:=WEIGHT:LENGTH:LENGTH ! ++:
rnd 2
=: BMI,AGE,BMI2,NAME,BMI3
AGE::=AGE wort
RGBRED      :=red leftat BMI
RGBDARKGREEN:=darkgreen leftat BMI2
RGBGREEN    :=green leftat BMI3
TITEL:="BMI averages total (red), per AGE (dark green), "
       + "per person and age (green)" leftat RGBRED
```

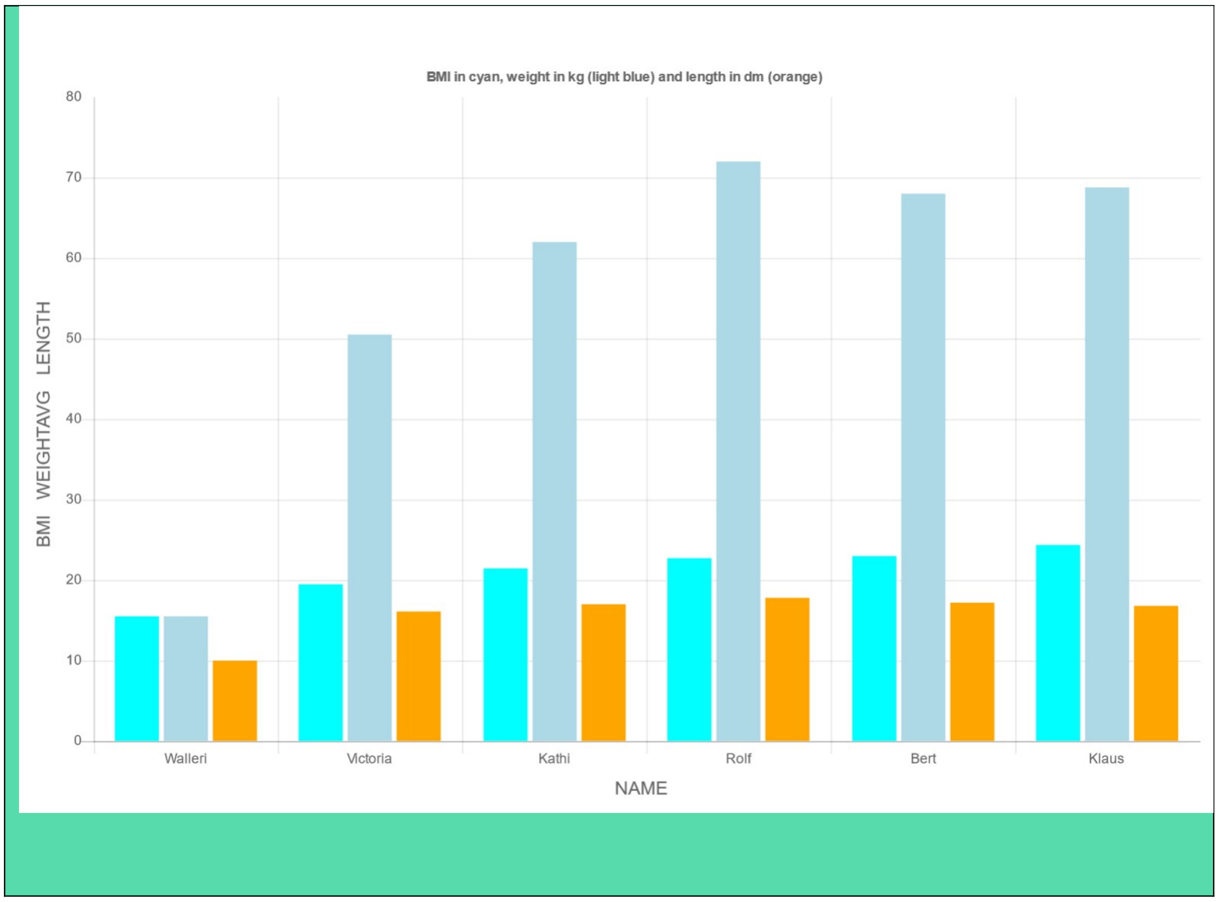
```
Result (struc.diagram bars)
```



Program 14.6: Compare the weights, lengths and BMI of all persons. Sort the persons by BMI.

```
aus weights.tabh
TITEL:="BMI in cyan, weight in kg (light blue) and length in dm (orange)"
gib TITEL,(BMI,NAME,WEIGHTAVG,LENGTH m)
    WEIGHTAVG:=WEIGHT ! ++:
    BMI:=WEIGHT:LENGTH:LENGTH ! ++:
LENGTH::=LENGTH*10
AGE::=AGE wort
RGBLIGHTBLUE:=lightblue leftat WEIGHTAVG
RGBORANGE   :=orange    leftat LENGTH
RGBCYAN     :=cyan      leftat BMI
```

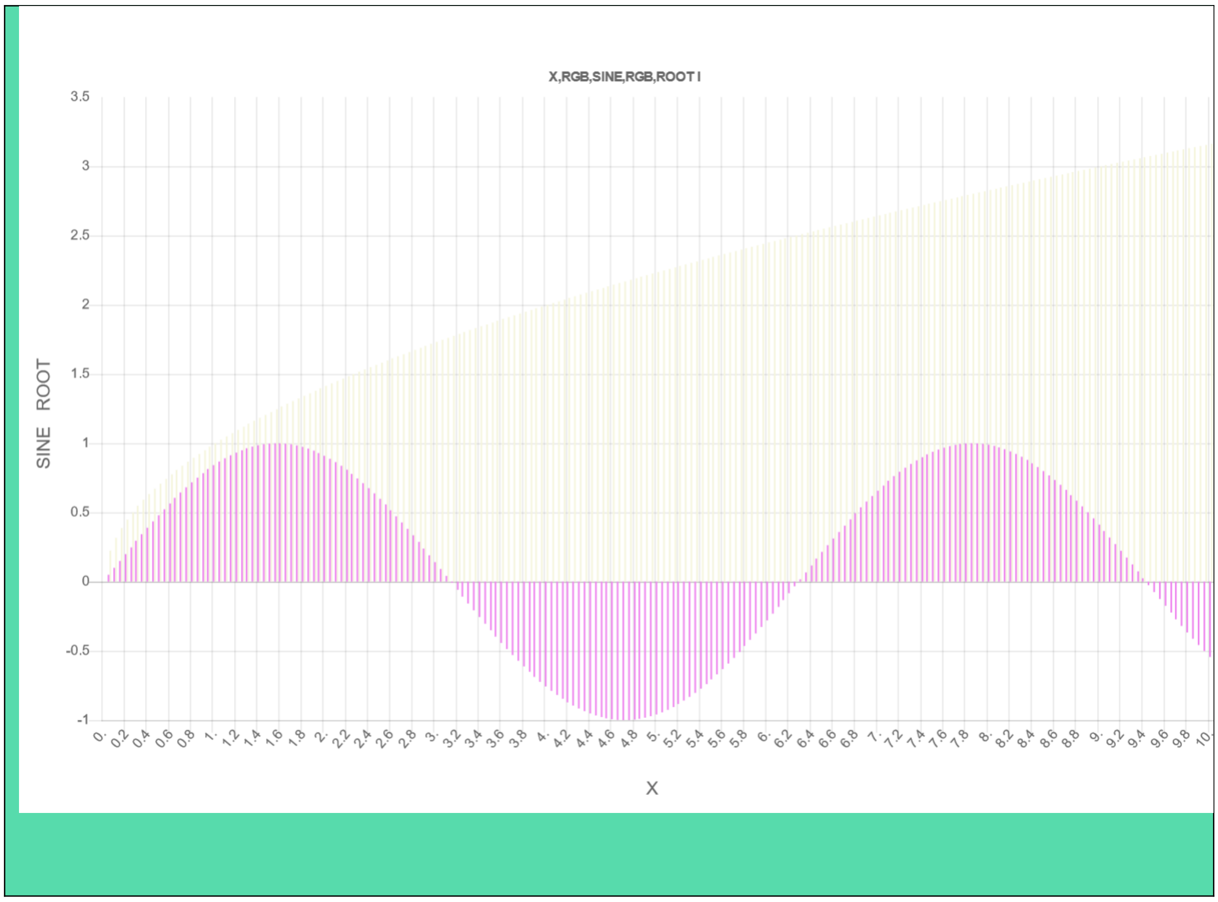
```
Result (struct.diagram bars)
```



Program 14.7: Represent 2 functions by bar graphs.

```
Xl:=0 ...10!0.05
SINE:=X sin
ROOT:=X sqrt
X::=X wort
RGB:=violet leftat SINE
RGB:=beige  leftat ROOT
```

```
Result (struc.diagram bars)
```

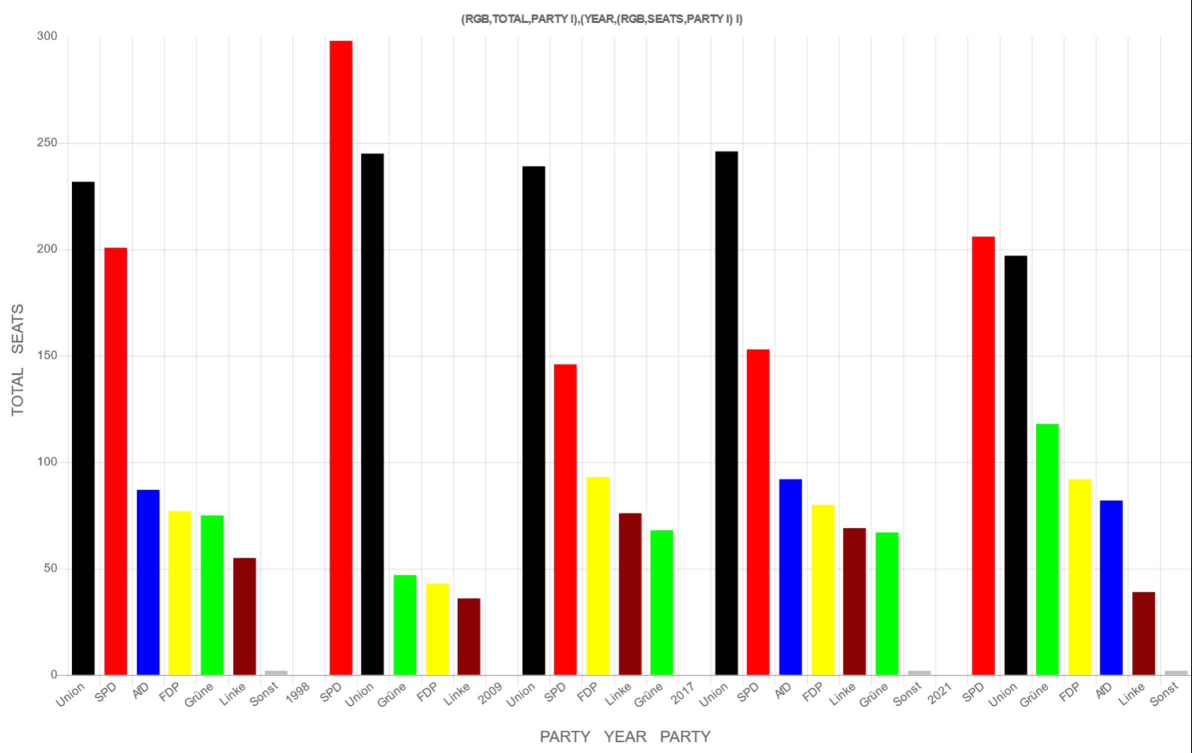


```
YEAR, (PARTY,SEATS l) l
1998    PDS     36
        SPD    298
        Grüne   47
        Union  245
        FDP     43
2009    Linke   76
        SPD    146
        Grüne   68
        Union  239
        FDP     93
2017    Linke   69
        SPD    153
        Grüne   67
        Union  246
        FDP     80
        AfD     92
        Sonst    2
2021    Linke   39
        SPD    206
        Grüne  118
        FDP     92
        Union  197
        AfD     82
        Sonst    2
```
elections.tab

Program 14.8: Visualize 4 election results and calculate the average number of votes of the 4 elections.

```
elections.tab
YEAR::=YEAR wort
PARTY::=Linke if PARTY="PDS" ! PARTY        # the PDS was renamed
gib  TOTAL,PARTY m-,(YEAR,(SEATS,PARTY m-)m)
     TOTAL:=SEATS! ++:
gib  TOTAL,PARTY l,(YEAR,(SEATS,PARTY l)l)
RGB:=red     if PARTY="SPD" !
     yellow  if PARTY="FDP" !
     darkred if PARTY=Linke !
     blue    if PARTY=AfD   !
     black   if PARTY=Union !
     green   if PARTY=Grüne !
     grey    leftat TOTAL SEATS
```

```
Result (struc.diagram bars)
```



```
Result without RGB values (tab)
TOTAL ,PARTY l, (YEAR ,(SEATS ,PARTY l) l)
231.75 Union     1998   298    SPD
200.75 SPD              245    Union
 87.   AfD               47    Grüne
 77.   FDP               43    FDP
 75.   Grüne             36    Linke
 55.   Linke     2009   239    Union
  2.   Sonst            146    SPD
                         93    FDP
                         76    Linke
                         68    Grüne
                 2017   246    Union
```

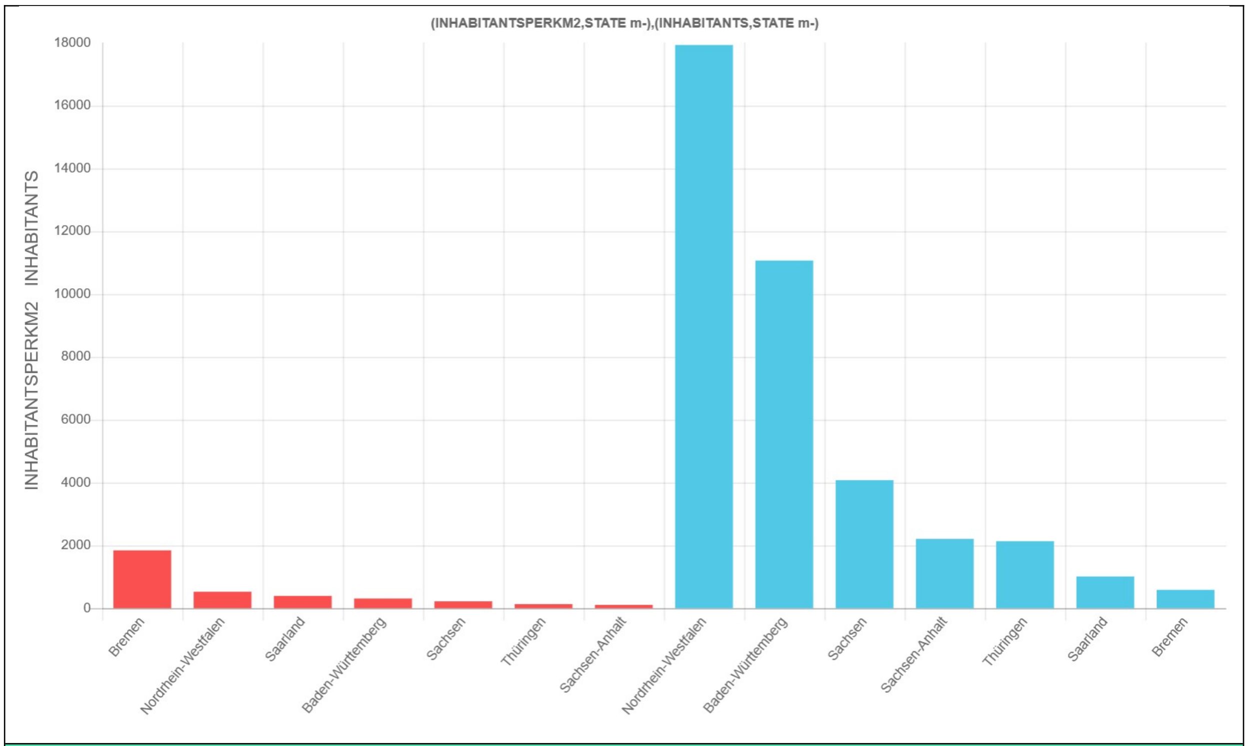
```
                  153     SPD
                   92     AfD
                   80     FDP
                   69     Linke
                   67     Grüne
                    2     Sonst
            2021  206     SPD
                  197     Union
                  118     Grüne
                   92     FDP
                   82     AfD
                   39     Linke
                    2     Sonst
```

```
states7.tab
STATE,              SHORT, AREA,    INHABITANTS, DEBTS l
Baden-Württemberg   BW     35751.46 11069          43.1
Bremen              HB       318.     586          23.8
Nordrhein-Westfalen NRW    34110.26 17932         174.5
Saarland            SL      2569.69  1012          13.4
Sachsen             SN     18449.99  4077           6.0
Sachsen-Anhalt      ST     20451.58  2208           3.5
Thüringen           TH     16172.5   2134           7.5
```

Program 14.9: Sort and visualize the states by population per area and by population. (Visualize 2 independent tables.)

```
states7.tab
INHABITANTSPERKM2:=INHABITANTS * 1'000 : AREA
gib INHABITANTSPERKM2,STATE m- , (INHABITANTS,STATE m-)
'3
rnd 0
```

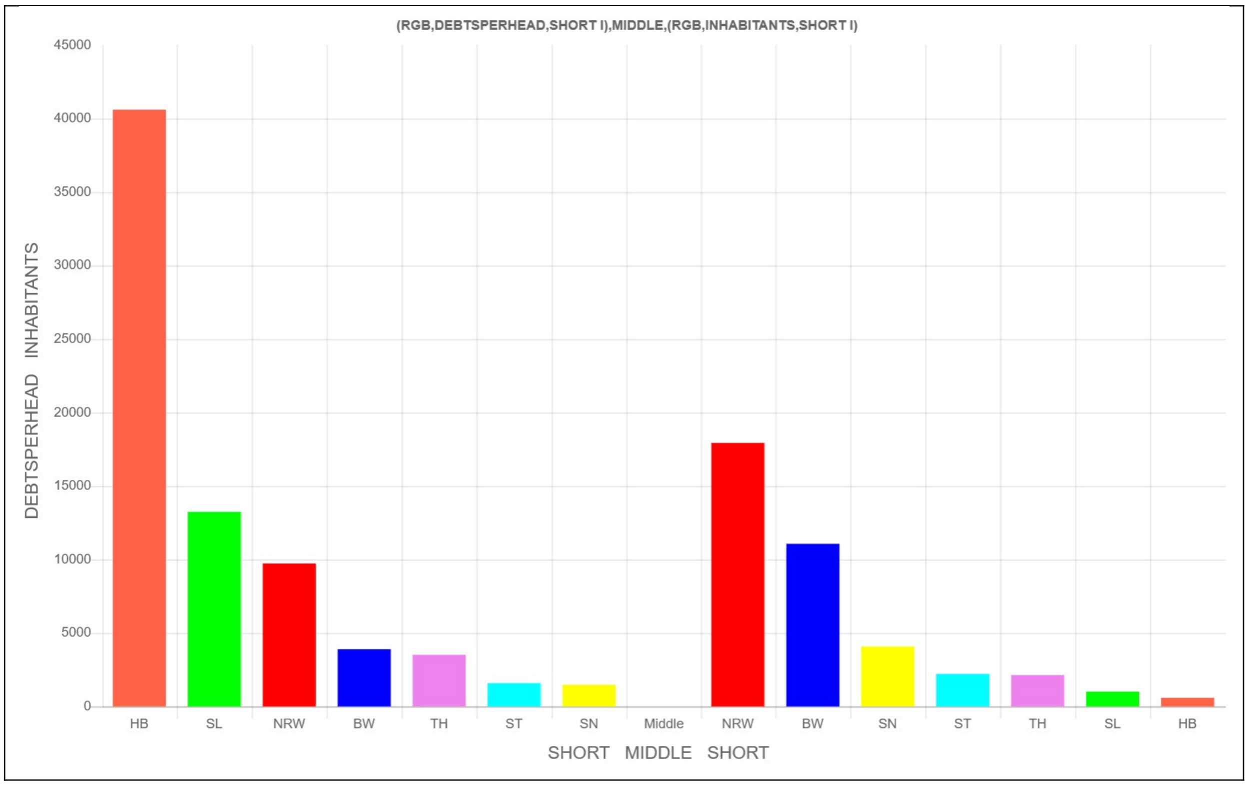
```
Result (struc.diagram bar )
```



```
Result (tab)
INHABITANTSPERAREA ,STATE m-,          (INHABITANTS ,STATE           m-)
1'843.              Bremen              17'932       Nordrhein-Westfalen
  526.              Nordrhein-Westfalen 11'069       Baden-Württemberg
  394.              Saarland             4'077       Sachsen
  310.              Baden-Württemberg    2'208       Sachsen-Anhalt
  221.              Sachsen              2'134       Thüringen
  132.              Thüringen            1'012       Saarland
  108.              Sachsen-Anhalt         586       Bremen
```

Program 14.10: Sort and visualize the states by population per area and by population, such that each states gets the same color in each of the diagrams. Divide the tables by additional space.

```
states7.tab
DEBTSPERHEAD:=DEBTS : INHABITANTS *1'000'000
MIDDLE:=Middle
gib DEBTSPERHEAD,SHORT m- ,MIDDLE,(INHABITANTS,SHORT m-)
gib DEBTSPERHEAD,SHORT l ,MIDDLE,(INHABITANTS,SHORT l)
RGB:= red    if SHORT="NRW" !
      blue   if SHORT="BW"  !
      yellow if SHORT="SN"  !
      green  if SHORT="SL"  !
      violet if SHORT="TH"  !
      tomato if SHORT="HB"  !
      cyan   leftat DEBTSPERHEAD INHABITANTS
Result (struc.diagram bars)
```


(RGB,DEBTSPERHEAD,SHORT I),MIDDLE,(RGB,INHABITANTS,SHORT I)
DEBTSPERHEAD INHABITANTS
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
HB
SL
NRW
BW
TH
ST
SN
Middle
NRW
BW
SN
ST
TH
SL
HB
SHORT MIDDLE SHORT

15 Multiple diagrams

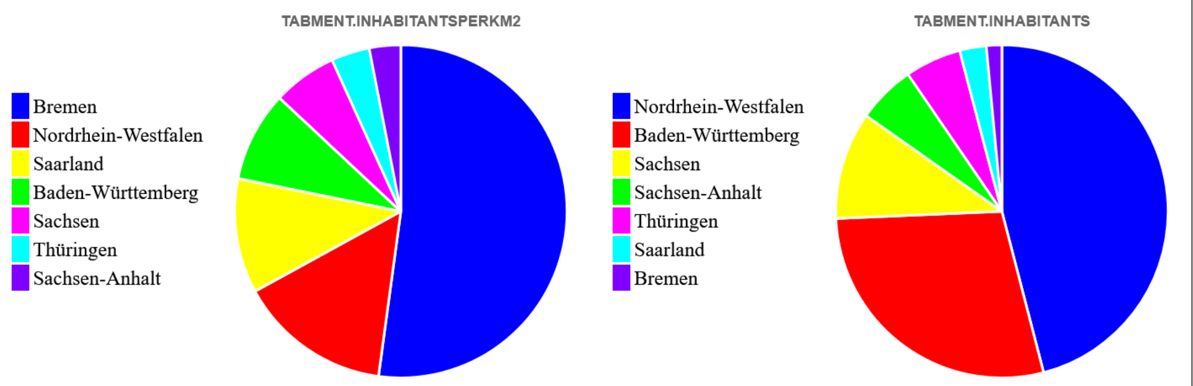
In the previous chapter we saw that a structured table can usually also be represented as a structured chart. Program 14.7 demonstrates that this also works for larger tables. However, pie charts quickly become confusing if a circle represents too many numbers.

Structured tables usually contain several sub-tables. These naturally contain fewer elements than the source table, so in this chapter each sub-table will be represented by an own diagram. With multiple diagrams, structured tables are visualized even more directly than with structured diagrams.

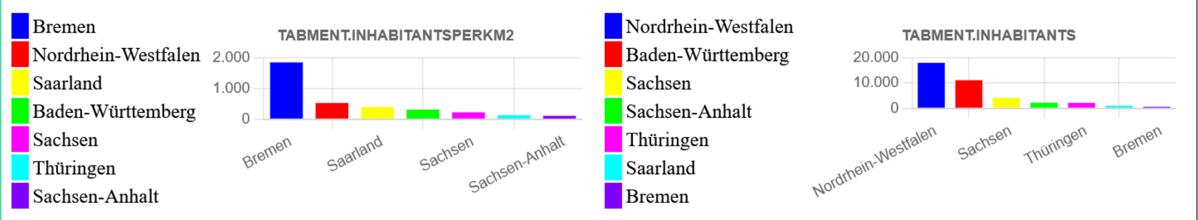
Program 15.1: Sort and visualize the states by population per square kilometer and by population. (Visualize 2 independent tables.) (Program 14.9)

```
states7.tab
INHABITANTSPERKM2:=INHABITANTS * 1'000 : AREA
gib INHABITANTSPERKM2,STATE m- , (INHABITANTS,STATE m-)
'3
rnd 0
```

Result (2 pie charts)



Result (2 bar charts)

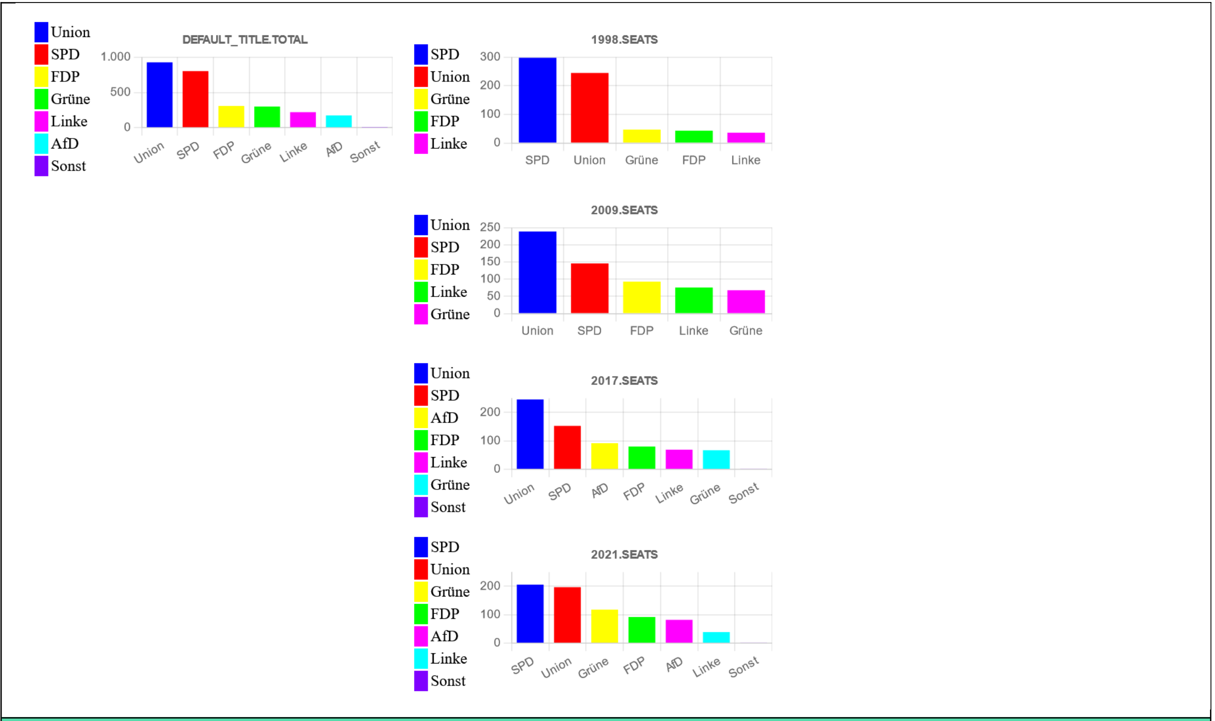


In program 14.9 the columns for inhabitants per square kilometer are much smaller than the columns for inhabitants. This problem disappears above with multiple charts.

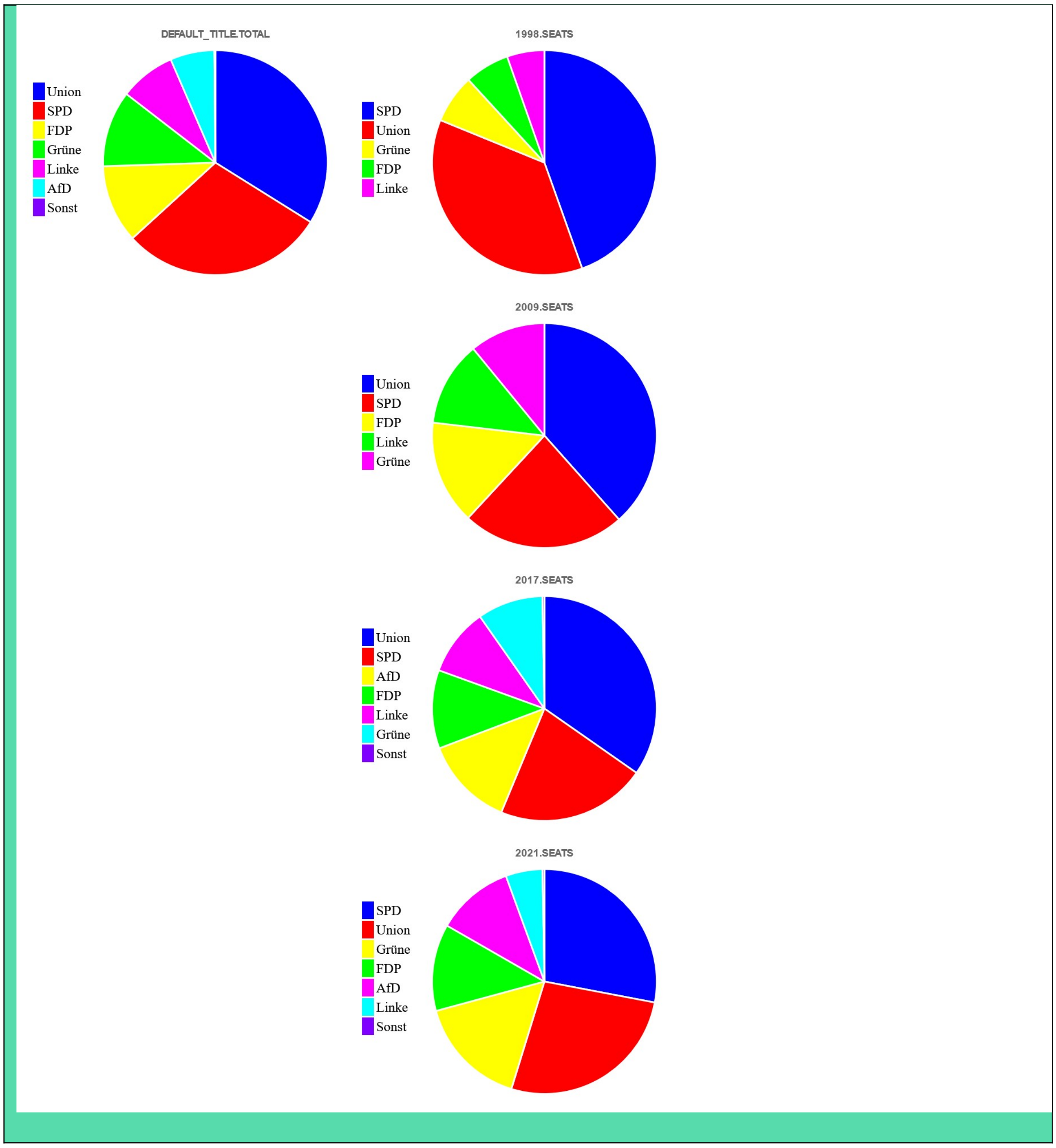
Program 15.2: as 14.10

```
elections.tab
YEAR::=YEAR wort
PARTY::=Linke if PARTY="PDS" ! PARTY
gib TOTAL,PARTY m-,(YEAR,(SEATS,PARTY m-)m) TOTAL:=SEATS ! ++
```

Result (5 bar charts)



Result (5 pie charts)



Note the difference between the above program and program 14.8. In 14.8 the sum of the four years is calculated and here the average is calculated. Therefore, the order of the parties in the total balance differs. The order does not change even if an "average" is calculated by division by 4. Here you can see how important it is that the end-user must be able to read the program in order to correctly understand the information received.

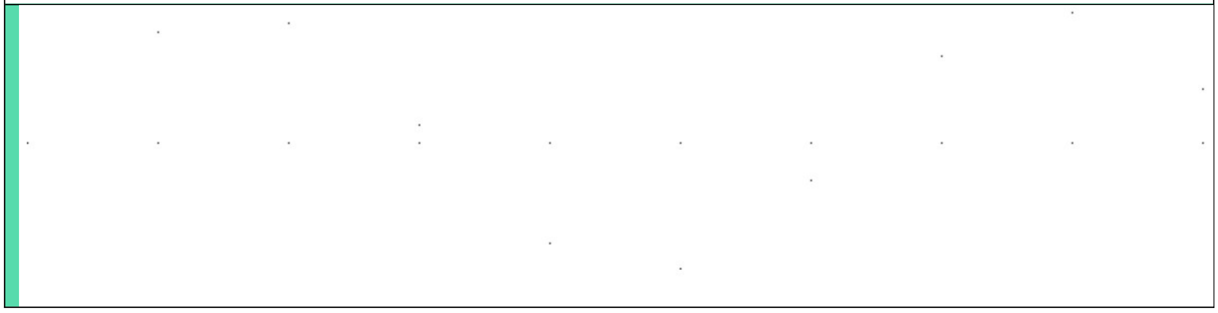
16 Image generation

Since o++o allows to generate numbers in a simple way, one can also generate whole images. For example, Xl:= 0 .. 4 generates the numbers 0 1 2 3 4. You can assign diagrams to these numbers, but to generate an image with o++o you need a list or a set of number pairs (X,Y). The point gets a color if there is an RGB (RED,GREEN,BLUE)-triple before the X or before the Y value:
(X, RGB, Y)
For RGB values o++o has 3 display options.
English color names:
red, silver, cyan, ...
Triples of integers between 0 and 255:
(255,0,0) (=red), (192,192,192) (=silver), (0,255,255) (=cyan)
Number triples between 0 and 1:
(1.,0.,0.) (=rot),(0.752941,0.752941;0.752941)(=silver),(0.,1.,1.) (=cyan)
We start with 2 functions, but initially define them only for 10 X values. You have to look closely to see the points:

Program 16.1: Create 10 points twice.

```
Xl:=0 ..9
Y :=X sin
Y0:=X*0
```

```
Result (image - new window)
```

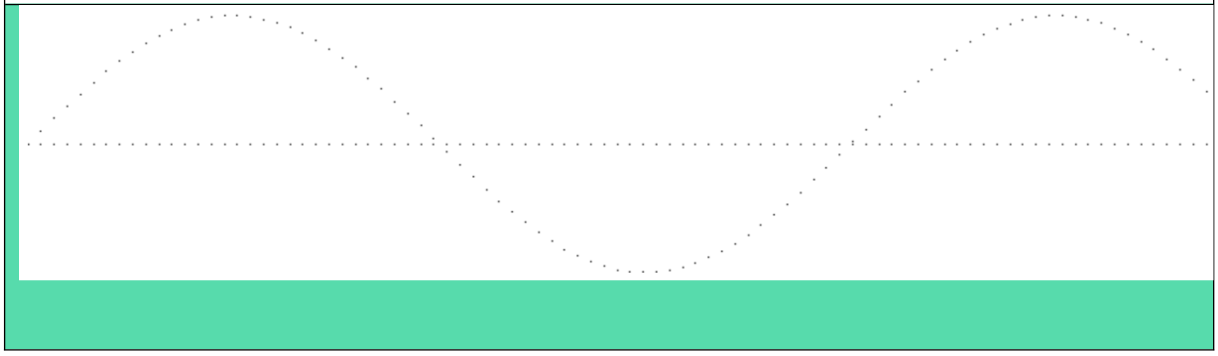


By introducing a step size of 0.1, the number of points is increased tenfold.

Program 16.2: Create 100 points twice.

```
Xl:= 0 ...9!0.1
Y:=X sin
Y0:=X*0
```

```
Result (image - new window)
```

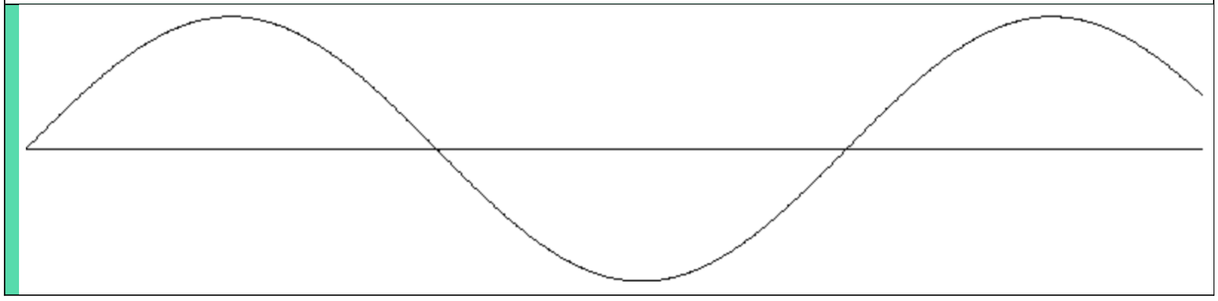


Now we add another 0 to the step size.

Program 16.3: Create 1000 points twice.

```
Xl:=0 ...9!0.01
Y :=X sin
Y0:=X*0
```

```
Result (image - new window)
```

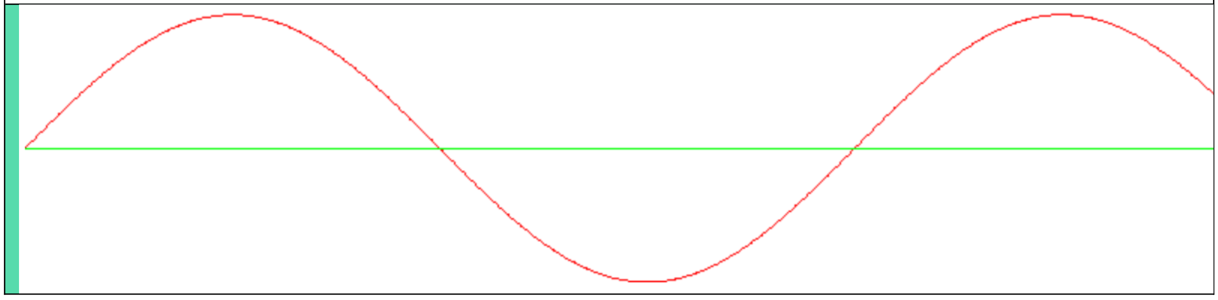


The sine function now becomes red and the X-axis green. The fact that a column name occurs twice (RGB) does not cause any problems at this point.

Program 16.4: Display 2 functions in color.

```
Xl:= 0 ...9!0.01
Y :=X sin
Y0:=X*0
RGB:=red   leftat Y
RGB:=green leftat Y0
```

```
Result (image - new window)
```



The fact that it is also possible to create "full images" is first shown by the German flag. You can see that all points that follow a color value are output in this color. Thus, each of the color values has to occur only once for generation of the below the German flag. The term pixel has lost its meaning here or must be redefined, because we allow structured tables.

Program 16.5: Generate the German flag

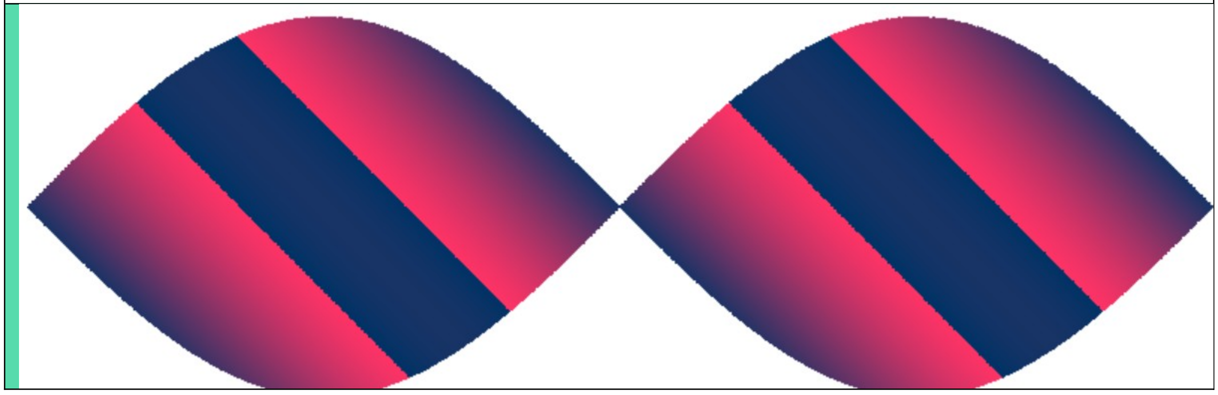
```
Xl:= 0 ...9!0.01
Yl:= 0 ...2!0.01 at X
=: $RECTANGLE
aus RGB:=gold
,$RECTANGLE
RGB:=red
,$RECTANGLE+(0,2)
RGB:=black
,$RECTANGLE+(0,4)
```

```
Result (image + new window)
```

Program 16.6: Design a bikini. Color the functions mirrored between sine and sine mirrored.

```
Xl:=pi * -1 ...pi!0.005
Yl:=X sin abs *-1 ...(X sin abs)! 0.005
RGB:= 0.1+(X+Y sin abs),0.2,0.4 leftat Y
```

```
Result (image + new window)
```



Appendix A: List of operations and keywords of o++o

Most of the known operations have an arity. The square root, for example, requires only one input value or argument - this is usually a number. Therefore, *sqrt* is unary and has the arity 1. In the o++o data model, the argument of *sqrt* can also be a list of numbers. Then the square root is taken from each of the numbers. The list is then considered to be **one** input value, even though it may contain ten or even ten thousand numbers. That is, *sqrt* remains unary even in this case.

In the o++o syntax the *sqrt* symbol must follow the argument (postfix). This means that no additional parentheses are required. It is not allowed to write sqrt([2 4 7]) in o++o. But instead, you can use
[2 4 7] sqrt
or shorter also
2 4 7 sqrt
.
In both cases you get the same result. You can even apply *sqrt* to any tabment.
Another example is the addition. The + operator is even better known than the root operation. It has arity 2, which means it requires two input values. The addition is binary. The application of the wrong number of arguments leads to a syntactical error and a corresponding error message.
3 +
as well as
3 4 +
lead to error messages.
In the term
3 + 4
3 is the first argument and 4 is the second input value. Again, a list or other tabment can be used as the first argument. The operation and the second argument are then applied to all elements of the list/table.
1 3 7 + 4
results in
5 7 11
Here and in many other operations the type of the result corresponds to the type of the first input table. So, the above result is also a list of numbers. Binary operations are always written between the two input tables in o++o. You can also say that they have to appear after the first input table like the unary operations. The same applies to ternary operations in o++o. "!" is used as a separator between the second and third input value.

Hadmersleben subtext 4!5
for example, has the result:
mersl
The first input value is "Hadmersleben". The second input value (4) specifies the position of the initial letter of the partial word and the third input value (5) specifies the desired length.
5 if X>3 ! 6
also requires 3 input values (here: the 5, a truth value, and the 6). If we replace X by 10, the condition is fulfilled and the improved "if-then-else" operation returns 5. For X=1, however, the value 6 results.

In the following, the input and output data are illustrated once again using typical examples.

(first) input tabment	**unary operation**		**output tabment**
pi	sin	results in	0.

1 4 9	sqrt	results in	1. 2. 3.
1 4 9	++:	results in	4.66666666666
123456789	'3	results in	123'456'789

First input table	**binary operation**	**second input table**		**output tabment**
7	+	8	results in	15
7.	+	8	results in	15.
1 5 3	+	4	results in	5 9 7
1 5 3	+	1.2	results in	2.2 6.2 4.2
1 5 3	+	3 7 8	results in	4 12 11
1 5 3	+	3 7	not defined	
7 9	divrest	3	results in	2,1 3,0

First input tabment	**ternary operation**	**second input tabment**		**third input tabment**		**output tabment**
"Georg Cantor"	subtext	1	!	5	results in	George
5	if	3=4 (no)	!	6	results in	6
1	...	2.9	!	0.5	results in	1. 1.5 2. 2.5
1	..x	2	!	4	Results for example in	1 2 1 2
1.	..x	2	!	4	Results for example in	1.59782294585 1.86688159101 1.78666803454 1.62531501586

At this point it should be noted that in many cases the result of the previous line counts as the first input value of an operation:

```
marks.tab
++:
```

gives the average of all numbers that occur in the first line marks.tab. marks.tab is the input of ++:
The program

```
xx.tab
+ 2
```

adds 2 to each number in table *xx.tab. xx.tab* is the first input table and 2 is the second. In an analogous way extracts
names.tab
subtext 3!4
from each text value (TEXT or WORT or ONR) of names.tab a text of length 4 starting at the third position. Here the ternary *subtext* operation has the input tabs names.tab, 3 and 4.
In an assignment or condition several operations can be applied one after the other. If all operations are unary (one input value), then each corresponding one-line term has the form

$\mathtt{tbt}\ \mathtt{op}_{11}\ \mathtt{op}_{12}\ \mathtt{op}_{13}\ \ldots\ \mathtt{op}_{1n}$

or more concretely:

```
1 2 3 sin abs sqrt ++text
```

If all operations are binary (two input values), the form is

$\mathtt{tbt}_1\ \mathtt{op}_{21}\ \mathtt{tbt}_2\ \mathtt{op}_{22}\ \mathtt{tbt}_3\ \mathtt{op}_{22}\ \mathtt{tbt}_4\ \ldots\ \mathtt{op}_{2n}\ \mathtt{tbt}_{n+1}$

or more concrete

`1 2 3 + 4 * 5 - 9`
Terms with only ternary operations are certainly rare. Here is just a constructed example:
`Magdeburg subtext 2!6 subtext 2!2`
results in gd
If brackets are set, they must be calculated first:
`abcdefghijk subtext 2!(2+3 )` results in `bcdef`
`abcdefghijk subtext 2!2+3` on the other hand results in `bc`
(`bc + 3` equals `bc`)
If you are not quite sure, you can put brackets as a precaution.
+ 3
is not a term, because the operation + has no first input value here. Therefore, an error message would appear. However, this would not be true if the above code were not on the first line. The result of the preceding line is then the first input value of `+ 3` is then the second input value.
In the following, the designations below are used:
num = ZAHL or PZAHL or RATIO or BAR (|) or BARl (stroke list)
nonum = TEXT or WORT or ONR
mixe = nonum **and** num occur in one column
tbt stands for any tabment type
For the types, we often specify only those that are also changed by the operation.

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
+	tbt + tbt→ tbt	1 1 3 + 2.1 results in 3.1 3.1 5.1 xy ab + de results in xyde abde	addition of numbers or connecting text
*	tbt * num→ tbt	2 3 5 * 2 results in 4 6 10	multiplication
-	tbt - num→ tbt	3 - 2 results in 1 11234 - 345 results in 10889	subtraction
:	tbt : num→ tbt	3:4 results in 0.75	division
_	_ -> tbt	Yl:= x y z X? := 1 if Y=y ! _ results in Y ,X? l x y 1 z	place holder
++	tbt ++→ num	2 3 6 ++ results in 11	sum
**	tbt **→ num	1 3 5 ** results in 15	product
--	tbt --→ num	20 5 4 -- results in 11	multiple subtraction
::	tbt ::→ num	64 2 2 :: results in 16	multiple division
++:	tbt ++:→ PZAHL	1 2 3 2 ++: results in 2.0	average
++1	tbt ++1→ZAHL	3 4 7 9 ++1 results in 4	count

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
++text	tbt ++text→ text	[ab cde fg] ++text results in abcdefg	connect to text
++textsep	TEXTl + +textsep "sep"→ TEXT	ab cde fg ++textsep ";" results in "ab;cde;fg"	combine to text with inclusion of a separator
,	tbt1,tbt2→ tbt	1 2,3 results in (.tab) ZAHLl,ZAHL 1 3 2	pairing
;	tbt1;tbt2→ tbt	2,3 *2 results in 4,6 2;3 *2 results in 2,6	also a pair formation. but ; separates sharper than ,
=	tbt1,tbt2→ BOOL	1 = 2 results in no	equality
<=	tbt1 <= tbt2→ BOOL	2 <= 2 results in si	less than or equal to
>=	tbt1 >= tbt2 → BOOL	2 >= 4 results in no	greater than or equal to

+coll	coll1 +coll
	coll2→ coll1
		{{1 2}} +coll {1}
		results in
		{{1 1 2 }}
			"set-theoretic" union

-coll	coll1 -coll coll2→ coll1	[2 4 3 2] -coll [2] results in 4 3 2	set difference
*coll	coll1 *coll coll2→ coll1	{1 2 3} *coll {4 5} results in ZAHL,ZAHL m 1 4 1 5 2 4 2 5 3 4 3 5	Cartesian product
:coll	coll1 :coll coll2→ coll1	{1 2 3} :coll [2 3 4] results in	intersection

Operation symbol	Notation: Input→ Result type	Examples	Meaning
		{2 3}	
*l	tbt *l ZAHL → tbt l	car *l 3 results in car car car or xx.tab *l 3	multiply an tabment by an integer to a list of elements
*mat	coll1 *mat coll2→ coll	(1,2) *mat [2 3] results in 8	matrix multiplication
-1mat	coll -1mat→ coll	<TAB! X1,X2,X3 l 1 0 2 0 2 0 0 0 8 !TAB> -1mat results in X1, X2, X3 l 1. -0. -0.25 -0. 0.5 -0. 0. -0. 0.125	inverse matrix
&	BOOL & BOOL -> BOOL	si & no results in no	conjunction (logical and)
&&	tbt &&→ BOOL	si,66,si && results in si	for all
|l	ZAHL |l -> BARl	5 |l results in | | | | |	transfer numbers into tally sheets (for kindergarten)
..	number1 .. num2 -> numl	1 .. 4 results in 1 2 3 4	from .. to generate numbers with step 1
...	number1 ... num2 ! num3→ numl	0 ... 0.6!0.2 results in 0. 0.2 0.4 0.6	from ... to ! step
..x	num1 ... num2 ! ZAHL -> numl	1 ..x 6!3 results in 5 3 2 (for example)	random numbers from ..x to ! cnt
'3	tbt '3→ tbt	1234567890 '3 results in 1'234'567'890	format in blocks of 3
'4	tbt '4→ tbt	12345.67898	format in blocks of 4

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
		'4 results in 1'2345.6789'8	
^ hoch	tbt ^ num→ tbt	4 ^ 1/2 results in 2. 10 *l 4 ^ (0 ..3) results in 1 10 100 1000	to the power of
abs	tbt abs→ tbt	-3 7 abs results in (tabh) 3 7	absolute amount
arctan	tbt arctan -> tbt	1 arctan results in 0.785398163397 (= pi:4)	arcus tangent
at		GROSS:=NET +% 19 at NET	place a new column to the right of the specified column
aus	tbt1 aus tbt2 → tbt2	aus rivers.tabh	(new) start of a program
comp	tbt name→ tbt	<TAB! NAME,FIRSTNAME,LOCATION Mill Paul Halle TAB> comp LOCATION results in LOCATION Halle (see also nth)	component
cos	num cos→ PZAHL	pi cos results in -1.	cosine
cross	tbt cross aggs -> tbt	[10 3] *mat (100,20,4) cross ++ results in (.tab) ZAHL ,ZAHL ,ZAHL ,SUM? l 1000 200 40 1240 300 60 12 372 1300 260 52 1612	generation of pivot tables
cut	tbt cut ZAHL -> tbt	123,Today cut 2 results in (.tab) WORTl, WORTl 12 To	cut elementary values to given length

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
		3 da y	
det	coll det→ coll	<TAB! X1,X2,X3 l 1 0 2 0 2 0 0 0 8 !TAB> det results in 16.	determinant
div	ZAHL div ZAHL→ ZAHL	11 div 5 results in 2	integer division
divrest	ZAHL divrest ZAHL→ pair	11 divrest 5 results in 2,1 (not 2.1)	integer division with remainder
e	e	e ^ 3 ln results in 3.	Euler's constant
first	tbt first -> tbt	1 3;4;7 8 9 first results in (tab) ZAHLl,ZAHL,ZAHLl 1 4 7	from each collection resp. elementary component preserve only the first element
gib	tbt1 gib schema → tbt2	aus students.tab gib FAC,(LOC,NAMEm)m	restructuring of a tabment, where a dtd, aggregations, and atomic schemes are allowed
giball	tbt1 giball scheme2 → tbt2	giball X | Y l List of all X and Y subtab segments (any depth); corresponds to ...//X|Y of XPath	extraction of all corresponding values; especially useful for recursive tabments
gibtop	tbt1 gibtop scheme2 → tbt2	gibtop Xl corresponds to: t/X: list of all X-subtabments of t, from the highest level of t.	extraction of the top values, only
if	term1 if cond ! term2 → tbt1	1 if 4=4 ! 2 results in 1	if with 3 input values

Operation symbol	Notation: Input→ Result type	Examples	Meaning
		1 if 4=3 ! 2 results in 2	
in	tbt1 in tbt2→ BOOL	"1 2 1" in "1 2" results in si "1 2 3" in "1 1 2" results no	every word of the left side is word of the right side
inmath	tbt1 inmath tbt2→ BOOL	[1 3] inmath [1 4 3] results in si 2 inmath {6 7 2} results in si	mathematical inclusion
keys	tbt1 keys tbt2→ tbt1	Xl:= 1 ..40 Y:=X*X gib X,Y m keys [7 34] results in (.tab) X, Y m 7 49 34 1156	efficient selection in sets or lists
keyslike	tbt1 keyslike tbt2→ tbt1	<TAB! NAME, LOC m Clara Oehna Claudia Dallgow Sophia Dallgow !TAB> keyslike ["*ia"] results NAME, LOC m Claudia Dallgow Sophia Dallgow	efficient selection in sets or lists with partial matching
last	tbt last -> tbt	1 2 4, 5 last results in (.tab) ZAHLl, ZAHL 4 5	from each collection resp. elementary component preserve only the last element
leftat		GR:=NET +% 19 leftat NET	place new column to the left of the specified column
like	term like "term?*" → BOOL	Hadmersleben like "?admers*" results in si '?': represents one letter	similar to

Operation symbol	Notation: Input→ Result type	Examples	Meaning
		'*': represents 0 or more letters	
linreg	tbt linreg→ num,num	<TAB! PRICE,SOLD l 20 0 16 3 15 7 16 4 13 6 10 10 !TAB> linreg results in 19.73214,-0.98214	linear regression
lists	tbt lists ZAHL → tbt l	Xl:= 1 2 lists 2 results in (.tabh) Xl l 1 1 1 2 2 1 2 2	generate a list of lists of specified length
ln	tbt ln→ PZAHL	e ln results in 1.	natural logarithm
log	tbt1 log tbt2→ PZAHL	100 log 10 results in 2.	general logarithm
lowercase	tbt lowercase → tbt	asdRRGee34 lowercase results in asdrrgee34	turn into lowercase letters
max	tbt max→ num	12.21,2,Hello max results in 12.21	maximum of all numbers
median	tbt median→ num	1 2 4.9 median results in 3.0	median
min	tbt min→ num	12.21,2,Hello min results in 2	minimum of all numbers
minus	tbt minus -> tbt	1 -2 4 minus results in (.tabh) -1 2 -4	negate any number

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
natsel	tbt natsel -> tbt	students.tab,exams.tab sel NAME=Ernst natsel after application of the condition “exams” also contains only exams from Ernst. The input-type remains unchanged.	natural selection (regarding common column names)
next		Xl:=1 2 3 Y:=100. next Y pred +% 10 at X results in X, Y l 1 100. 2 110. 3 121.	recursive assignment
nextonr		Xm:={1 1.1 1.1.2 2} Y:=1 nextonr Y pred + 10 at X results in X, Y m 1 1 1.1 11 1.1.2 21 2 1	next for onr-recursion
no	no→ BOOL	no or si results in si	truth value false corresponds to the answer no (Spanish no)
not	BOOL not→ BOOL	si not results in no	negation
nth	tbt nth ZAHL → tbt'	1 3 5 nth 2 results in 3	nth component
nthpred	Name nthpred ZAHL→ term	Xl:= 1 2 3 4 Y:= X nthpred 2 results in X,Y? l 1 2 3 1 4 2	n-th predecessor
nthsucc	tbt nthsucc ZAHL→ tbt'	Xl:=2 4 7 4 3 4 4 sel X nthsucc 2=4 results in (tabh) 4 4 3	n-th successor

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
nthzahl	tbt nthzahl ZAHL→ tbt'	"2023.03.26" nthzahl 3 results in 26	nth number in a text
onr	tbt onr→ tbt	1 3 5.2 "4.5.5" onr results in (.tabh): 1 3 5.2 4.5.5	Conversion to o++o number
onrs	tbt onrs name ! element→ tbt'	<TAB! PART, SUBPARTm m car motor body motor reel screw !TAB> onrs car results in PART,(OTTONR,SUBPART m)l car 1 body 2 motor 2.1 reel 2.2 screw	generates o++o numbers in a table; this is an important component of BOM explosion.
or	BOOL or BOOL -> BOOL	1=1 or 1=2 results in si	disjunction (logical or)
or2	coll1 or2 → BOOL	1=2,no or2 results in no	it exists

permutations	list permutations -> listl	2 4 9
		permutations
		results in (.tabh)
		ZAHLl l
		2 4 9
		2 9 4
		4 2 9
		4 9 2
		9 2 4
		9 4 2
			"permutations" is an abbreviation for the program:
			Xl:= 2 4 9
			lists 3
			sel Xm= {2 4 9}

pi	pi→ PZAHL	CIRCULAR_AREA:=R*R*pi	circle number
poly	num poly list→ num	3 poly [1 2 3] results in 18	polynomial
pos	Name pos→ ZAHL	sel X pos < 10	position
pos-	Name pos-→ ZAHL	sel X pos- > 5	position from behind

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
pred	Name pred→ term	X:=100 next X pred *1.03	predecessor
proj	tbt NAMES -> tbt	<TABH! X,Ym m 1 2 3 4 5 !TABH> proj Y results in (.tabh) Ym m 5 2 3	omitting columns
proj-	tbt proj- NAMES → tbt	<TABH! X,Ym m 1 2 3 4 5 !TABH> proj- Y results in (.tab) Xm 1 4	omitting columns
pzahl	tbt pzahl → PZAHL	1/5 6 9.7 pzahl results in (.tabh) 0.2 6. 9.7	conversion to a PZAHL (float)
pzahl1de	tbt pzahl1de→ PZAHL	"Today I get 356,88 euros and not 66.8 ." pzahl1de results in 356.88	first German “Comma number” of a text
rat	ZAHL rat ZAHL→ RATIO	<TAB! X,Yl l 1 2 3 TAB> Z:= X rat Y results in X,(Y, Z l) l 1 2 1/2 3 1/3	conversion of two integers into one RATIO number
ratio	num ratio→ RATIO	1/5 6 9.7 ratio results in (.tabh) 1/5 6/1 97/10	conversion to rational number
rename	tbt	rename X!Y	renaming column names

Operation symbol	Notation: Input→ Result type	Examples	Meaning
	rename Name1 ! name2→ tbt'		
rest	ZAHL rest ZAHL→ ZAHL	13 rest 5 results in 3	remainder of integer division
rnd	PZAHL rnd ZAHL→ PZAHL	17.678 3.45 zz 8 rnd 1 results in 17.7 3.5 zz 8	round
route	tbt route→ tbt	<TAB! X,Y m 0 0 1 1 0 1 !TAB> route generates 2 lines from (0,0) to (1,1) and from (1,1) to (0,1)	generate a straightline sequence from point sequence
satzl	TEXT satzl TEXTl	"It's great. Great. Tomorrow we celebrate." satzl results in (.tabh) SATZl It's great. Great. Tomorrow we celebrate.	list of sentences
seg	Name seg→ term	grandson.tabh sel Oehna in NAME seg or X seg ++: average of all numbers of the segments, containing X	segment
sel	tbt1 sel cond → tbt	rivers.tabh sel LENGTH >800	selection
sel-	tbt1 sel- cond → tbt	sel- LOC=Magdeburg sel- Magdeburg sel-: without the specified struples	selection
sepl	constant list, useful for an additional	"." ";" "," "|" "!" "?" "(" ")" "@" "#" "\n" "-" " "	all separators useful in cut operations

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
	cut operations		
si	si → BOOL	si & no results in no	truth value true (answer yes (=si))
sin	PZAHL sin → PZAHL	3.14159 sin results in 2.65358979335e-06	sine function
sqrt	num sqrt→ PZAHL	4 sqrt results in 2.	square root
mad streu	tbt mad → PZAHL	[1 2 5 3 5 1] mad results in 1.5	scattering
subtext	text subtext ZAHL ! ZAHL→ TEXT	aBCdE subtext 2 ! 3 results in BCd	subtext (substring)
subtext2	text subtext2 text ! text→ TEXT	aBCdEfgh subtext2 "B"!fg results in CdE	partial text of the first text that lies between the other two given texts.
succ	Name succ→ term	MARKl:= 3 1 2 1 sel MARK >MARK succ results in MARKl 3 2	Successor
tag	tbt tag NAME! scheme → tbt'	LOCATION:=Magdeburg STREET:=Beims tag ADDRESS! LOCATION,STREET results (metadata) TABMENT ! ADDRESS ADDRESS ! LOCATION,STREET LOCATION ! WORT STREET ! WORT	enclose data of a schema with a tag
tag0	tbt tag0 name→ tbt'	11 13 tag0 XX results (ment) <XX> 11 13 </XX>	put a tag around the entire tabment
tan	num tan→ PZAHL	3.14 tan results in -0.00159265493641	tangent function

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
text	mixe text→ TEXT	3.14 ttt 8 text results in TEXTl 3.14 ttt 8	transform any elementary type to TEXT.
textend	tbt textend ZAHL→ TEXT	asdfgh text end 4 results fgh abcde textend -2 results in de	subtext counted from back
textindex	text textindex text→ ZAHL	"Today is Tuesday." textindex Tu results in ZAHL 10	Position
time	time→ PZAHL	time results in: 1.557021 (for example)	system time (only the difference between two such times is significant for efficiency considerations)
total	tbt total aggs -> tbt	facs.tab total ++,++: results in (.tab) FAC ,DEAN ,BUDGET , STUDCAPACITY l Art Sitte 2000 600 Infor Reichel 10000 500 Math Dassow 1000 200 Philo Hegel 1000 10 Sport Streich 8000 150 sum sum 22000 1460 avg avg 4400. 292.	One or more aggregations at the end of each collection
trim	text trim→ text	" Hi o++o " trim Results in (ment) <TABM>Hi o++o</TABM>	remove spaces at the back and front
tup	NAME tup→ tupel	grandson.tabh sel German in NAME tup	a whole tuple
untag0	tbt untag0→ tbt'	X:=1 untag0 results in ZAHL 1	remove the outermost tag
uppercase	text uppercase→ text	1.2,aW uppercase results in (.tab) PZAHL,WORT 1.2 AW	convert to uppercase

Operation symbol	**Notation: Input→ Result type**	**Examples**	**Meaning**
variance	tbt variance→ PZAHL	[1 2 4 6] variance results in 4.91666666667	variance
vlists	tbt vlists ZAHL→ tbt l	variable-length lists; the operation generates the same as "lists" except that all shorter lists also appear in the result.	variable length lists
wort	tbt wort→ wort	"I'm good.So are you." wort results in WORT I_am_good.You_too.	convert to words
zahl	num zahl→ ZAHL	"12" zahl results in 12 3.14 zahl results in 3	Conversion into integers
zahltrip	text zahltrip -> Triples of nums	DAY,MON,YEAR:= 26.03.1963 zahltrip results in DAY,MON,YEAR 26 3 1963	the first 3 numbers of a text
zahlratio	RATIO zahlratio -> ZAHL,RATIO	33/7 zahlratio results in 4 5/7	convert to integer part and real fraction
zahl1de	text zahl1de → ZAHL	"Today I get 66,356 11 euros" zahl1de results in 66	extract first integer from a German text

Appendix B: List of o++o color names

```
"aliceblue",(0.941176470588,0.972549019608,1.);
"antiquewhite",(0.980392156863,0.921568627451,0.843137254902);
"aquamarine",(0.498039215686,1.,0.83137254902);
"azure",(0.941176470588,1.,1.);
"beige",(0.960784313725,0.960784313725,0.862745098039);
"bisque",(1.,0.894117647059,0.76862745098);
"black",(0.,0.,0.);
"blanchedalmond",(1.,0.921568627451,0.803921568627);
"blue",(0.,0.,1.);
"blueviolet",(0.541176470588,0.16862745098,0.886274509804);
"brown",(0.647058823529,0.164705882353,0.164705882353);
"burlywood",(0.870588235294,0.721568627451,0.529411764706);
"cadetblue",(0.372549019608,0.619607843137,0.627450980392);
"chartreuse",(0.498039215686,1.,0.);
"chocolate",(0.823529411765,0.411764705882,0.117647058824);
"coral",(1.,0.498039215686,0.313725490196);
"cornflowerblue",(0.392156862745,0.58431372549,0.929411764706);
"cornsilk",(1.,0.972549019608,0.862745098039);
"cyan",(0.,1.,1.);
"darkgoldenrod",(0.721568627451,0.525490196078,0.043137254902);
"darkgreen",(0.,0.392156862745,0.);
"darkkhaki",(0.741176470588,0.717647058824,0.419607843137);
"darkolivegreen",(0.333333333333,0.419607843137,0.18431372549);
"darkorange",(1.,0.549019607843,0.);
"darkorchid",(0.6,0.196078431373,0.8);
"darkred",(0.5450,0.,0.);
"darksalmon",(0.913725490196,0.588235294118,0.478431372549);
"darkseagreen",(0.560784313725,0.737254901961,0.560784313725);
"darkslateblue",(0.282352941176,0.239215686275,0.545098039216);
"darkslategray",(0.18431372549,0.309803921569,0.309803921569);
"darkturquoise",(0.,0.807843137255,0.819607843137);
"darkviole",(0.580392156863,0.,0.827450980392);
"deeppink",(1.,0.078431372549,0.576470588235);
"deepskyblue",(0.,0.749019607843,1.);
"dimgrey",(0.411764705882,0.411764705882,0.411764705882);
"dodgerblue",(0.117647058824,0.564705882353,1.);
"firebrick",(0.698039215686,0.133333333333,0.133333333333);
"floralwhite",(1.,0.980392156863,0.941176470588);
"forestgreen",(0.133333333333,0.545098039216,0.133333333333);
"gainsboro",(0.862745098039,0.862745098039,0.862745098039);
"ghostwhite",(0.972549019608,0.972549019608,1.);
"gold",(1.,0.843137254902,0.);
"goldenrod",(0.854901960784,0.647058823529,0.125490196078);
"green",(0.,1.,0.);
"greenyellow",(0.678431372549,1.,0.18431372549);
"grey",(0.745098039216,0.745098039216,0.745098039216);
"honeydew",(0.941176470588,1.,0.941176470588);
```

```
"hotpink",(1.,0.411764705882,0.705882352941);
"indianred",(0.803921568627,0.360784313725,0.360784313725);
"ivory",(1.,1.,0.941176470588);
"lavender",(0.901960784314,0.901960784314,0.980392156863);
"lavenderblush",(1.,0.941176470588,0.960784313725);
"lawngreen",(0.486274509804,0.988235294118,0.);
"lemonchiffon",(1.,0.980392156863,0.803921568627);
"lightblue",(0.678431372549,0.847058823529,0.901960784314);
"lightcoral",(0.941176470588,0.501960784314,0.501960784314);
"lightcyan",(0.878431372549,1.,1.);
"lightgoldenrod",(0.933333333333,0.866666666667,0.509803921569);
"lightgray",(0.827450980392,0.827450980392,0.827450980392);
"lightpink",(1.,0.713725490196,0.756862745098);
"lightsalmon",(1.,0.627450980392,0.478431372549);
"lightseagreen",(0.125490196078,0.698039215686,0.666666666667);
"lightskyblue",(0.529411764706,0.807843137255,0.980392156863);
"lightslateblue",(0.517647058824,0.439215686275,1.);
"lightslategray",(0.466666666667,0.533333333333,0.6);
"lightsteelblue",(0.690196078431,0.76862745098,0.870588235294);
"lightyellow",(1.,1.,0.878431372549);
"limegreen",(0.196078431373,0.803921568627,0.196078431373);
"linen",(0.980392156863,0.941176470588,0.901960784314);
"ltgoldenrodyello",(0.980392156863,0.980392156863,0.823529411765);
"magenta",(1.,0.,1.);
"maroon",(0.690196078431,0.188235294118,0.376470588235);
"mediumaquamarine",(0.4,0.803921568627,0.666666666667);
"mediumblue",(0.,0.,0.803921568627);
"mediumorchid",(0.729411764706,0.333333333333,0.827450980392);
"mediumpurple",(0.576470588235,0.439215686275,0.858823529412);
"mediumseagreen",(0.235294117647,0.701960784314,0.443137254902);
"mediumslateblue",(0.482352941176,0.407843137255,0.933333333333);
"mediumturquoise",(0.282352941176,0.819607843137,0.8);
"mediumvioletred",(0.780392156863,0.0823529411765,0.521568627451);
"medspringgreen",(0.,0.980392156863,0.603921568627);
"midnightblue",(0.0980392156863,0.0980392156863,0.439215686275);
"mintcream",(0.960784313725,1.,0.980392156863);
"mistyrose",(1.,0.894117647059,0.882352941176);
"moccasin",(1.,0.894117647059,0.709803921569);
"navajowhite",(1.,0.870588235294,0.678431372549);
"navyblue",(0.,0.,0.501960784314);
"oldlace",(0.992156862745,0.960784313725,0.901960784314);
"olivedrab",(0.419607843137,0.556862745098,0.137254901961);
"orange",(1.,0.647058823529,0.);
"orangered",(1.,0.270588235294,0.);
"orchid",(0.854901960784,0.439215686275,0.839215686275);
"palegoldenrod",(0.933333333333,0.909803921569,0.666666666667);
"palegreen",(0.596078431373,0.98431372549,0.596078431373);
"paleturquoise",(0.686274509804,0.933333333333,0.933333333333);
"palevioletred",(0.858823529412,0.439215686275,0.576470588235);
"papayawhip",(1.,0.937254901961,0.835294117647);
"peachpuff",(1.,0.854901960784,0.725490196078);
"peru",(0.803921568627,0.521568627451,0.247058823529);
"pink",(1.,0.752941176471,0.796078431373);
```

```
"plum",(0.866666666667,0.627450980392,0.866666666667);
"powderblue",(0.690196078431,0.878431372549,0.901960784314);
"purple",(0.627450980392,0.125490196078,0.941176470588);
"red",(1.,0.,0.);
"rosybrown",(0.737254901961,0.560784313725,0.560784313725);
"royalblue",(0.254901960784,0.411764705882,0.882352941176);
"saddlebrown",(0.545098039216,0.270588235294,0.0745098039216);
"salmon",(0.980392156863,0.501960784314,0.447058823529);
"sandybrown",(0.956862745098,0.643137254902,0.376470588235);
"seagreen",(0.180392156863,0.545098039216,0.341176470588);
"seashell",(1.,0.960784313725,0.933333333333);
"sienna",(0.627450980392,0.321568627451,0.176470588235);
"silver",(0.898039215686, 0.898039215686, 0.898039215686);
"skyblue",(0.529411764706,0.807843137255,0.921568627451);
"slateblue",(0.41568627451,0.352941176471,0.803921568627);
"slategrey",(0.439215686275,0.501960784314,0.564705882353);
"snow",(1.,0.980392156863,0.980392156863);
"springgreen",(0.,1.,0.498039215686);
"steelblue",(0.274509803922,0.509803921569,0.705882352941);
"tan",(0.823529411765,0.705882352941,0.549019607843);
"thistle",(0.847058823529,0.749019607843,0.847058823529);
"tomato",(1.,0.388235294118,0.278431372549);
"turquoise",(0.250980392157,0.878431372549,0.81568627451);
"violet",(0.933333333333,0.509803921569,0.933333333333);
"violetred",(0.81568627451,0.125490196078,0.564705882353);
"wheat",(0.960784313725,0.870588235294,0.701960784314);
"white",(1.,1.,1.);
"whitesmoke",(0.960784313725,0.960784313725,0.960784313725);
"yellow",(1.,1.,0.);
"yellowgreen",(0.603921568627,0.803921568627,0.196078431373)
```