o++o auf 41 Seiten

(Stand 05.01.2022)

Klaus Benecke

Inhalt

1 Vision

Es gibt die 4 Einzeldaten-Grundoperationen Addition, Subtraktion, Multiplikation und Division mit nahezu standardisierten Bezeichnungen `+, -, × (*)` bzw. `: (÷)`. Außerdem gibt es viele weitere Operationen wie `sin, cos, log,` usw. Aufgrund der fortschreitenden Digitalisierung benötigen wir auch leistungsfähige Massendatenoperationen mit Standards. Die Menge der mengentheoretischen Operationen wie Schnittmenge (∩), Vereinigungsmenge (∪), Mengendifferenz (\ oder -) wurden um das kartesische Produkt (×), den Join (|×|), die Projektion (π) und Auswahloperationen (Selektionen) (σ) erweitert. Diese Operationen sind in SQL - der Standarddatenbanksprache für relationale Datenbanken - verborgen. Der größte Teil der heutigen Daten werden in relationalen Systemen wie ORACLE, MYSQL, HANA, DB2, MariaDB, ... gespeichert. Das Relationale Modell wurde von dem britischen Mathematiker E. F. Codd erfunden. Da das Relationale Modell nicht leistungsfähig genug war, um die Anforderungen der industriellen Anwendungen zu erfüllen, erweiterten die Informatiker das Modell um Multi-Sets, einfache Aggregatfunktionen wie sum, avg, usw., um groupby, having, Sortieroperationen usw. . Trotz dieser Erweiterungen kann man strukturierte Tabellen nicht mit SQL verarbeiten. Es gibt aber sehr einfache strukturierte Tabellen wie

```
FACH,  NOTEl l
Mathe  1 2 2 3 1
Physik 2 1 4 1 3 4
```

, die von Kindern verstanden werden können. Hier haben wir für jedes Fach eine Liste (l) von Noten. Die gesamte Tabelle besteht aus einer Liste (das letzte `l`) von (`FACH,NOTEl`)-Paaren. SQL und sogar EXCEL sind nicht in der Lage, solche Schemata wie (`FACH,NOTEl l`) oder höher strukturierte Tabellen zu verarbeiten. Ferner kennt SQL keine strukturierten Texte. Daher glauben wir, dass SQL sein ursprüngliches Ziel - eine Endnutzersprache zu werden - nie erreichen wird. Wir brauchen also einen neuen Standard für Datenbanken und Dokumentensammlungen.
Unsere Vision ist, dass o++o dieser Standard für die Abfrage von relationalen Datenbanken, Big-Data-Systemen und sogar Systemen wird, die riesige Mengen von Dokumenten verwalten. Es soll dem Endnutzer damit dieses mächtige Mittel der strukturierten Tabelle zur Verfügung gestellt werden.
Facebook und Co. arbeiten daran Nachrichten so aufzubereiten, dass sie vor allem Emotionen ansprechen, als wäre die gesellschaftliche Situation nicht schon ausreichend aufgeheizt. Unser Ansatz ist ein anderer. Wir arbeiten daran dem Endnutzer Werkzeuge bereitzustellen, um seine rationale Urteilskraft mit Hilfe des Computers zu stärken. Dafür muss er auf umfangreiche, vertrauenswürdige, öffentlich zugängliche Informationen in Form von vielgestaltigen großen Tabellen und Dokumenten ähnlich der Wikipedia zugreifen können. Der Einzelne soll diese eigenständig auswerten können. Nur so kann er sich selbst ein Urteil bilden.
Auch wenn diese Sprache so einfach wie möglich ist, erfordert sie im Gegensatz zum „Facebookansatz“ einen gewissen Lernaufwand.

2 Entwurfskriterien für eine Endnutzersprache

1. Eine **mathematische Fundierung** ist erforderlich.
2. **Methodisch-didaktische und pragmatische Fragen** stehen zunächst **vor Effizienz-problemen.**
3. Die Endnutzerprogramme sollen kurz und gut lesbar sein, ebenso die wichtigsten Schlüsselworte.
4. **Verzicht auf Schleifen** (for, while). Schleifen führen häufig zu schwer lesbaren und schwer änderbaren Programmen. **Allgemeine Rekursion** erfordert ein relativ hohes Abstraktions-niveau und damit einen hohen Lernaufwand; auch darauf sollte verzichtet werden. Daraus resultiert, dass die Programme einfach von oben nach unten und von links nach rechts geschrieben und abgearbeitet werden sollten. Das erfordert leistungsfähige Operationen.
5. Die Endnutzersprache soll **universell anwendbar** sein. Sie muss nicht nur für Relationen (flache einfache Tabellen), sondern auch für strukturierte Tabellen und strukturierte Dokumente nutzbar sein. Sie soll nicht nur für Anfragen an die verschiedensten Systeme, sondern auch für vielfältige Berechnungen geeignet sein.
6. Um im Schulunterricht einsetzbar zu sein, soll sie **die verschiedene**n **mathematische**n **Teilgebiete unterstützen**, sowie Nutzen für die anderen Fächer bieten.
7. Die Endnutzersprache soll so **mächtig** sein, dass sie andere Systeme und Sprachen wie Tabellenkalkulation, XQuery, XPath und SQL ersetzen kann.
8. Aus Endnutzersicht soll es ein einheitliches System mit **einheitlicher Syntax** (Schreibweise) für die Verarbeitung von Massendaten geben, genau wie die Operationen der Einzeldatenverarbeitung (+ - * : sqrt sin …) standardisiert sind.

3 Berechnungen mit o++o – mehr als ein Taschenrechner

o++o ist anwenderfreundlich konzipiert. Es besitzt eine einfache und einprägsame Syntax sowie kurze Programme. Seine Rechenoperationen folgen der Logik des Kopfrechnens und laufen daher wie auf einem einfachen Taschenrechners ab.
Die folgenden Programme sollte man unter ottops.de ausprobieren, da man Programmieren kaum ohne eigenes Testen erlernen kann.
o++o, genauer ottoPS (ottoProgrammierSprache), ist nicht nur ein wesentlich erweiterter Taschenrechner. o++o erlaubt auch Anfragen an strukturierte TABellen und DokuMENTe (Tabmente). o++o ist damit auch geeignet, Datenbankanfragen zu ermöglichen und später Basis bestimmter Suchmaschinen zu werden. o++oPS ist eine Endnutzerprogrammiersprache, die Wiederholgruppen einfach verwalten kann. Sie erlaubt es, Anfragen an Tabellen und Dokumente besonders einfach zu formulieren.

Programm 3.1: Berechne die vierte Wurzel aus 16.
```
16 sqrt sqrt
```

Ergebnis (nach dem Klicken von tab):
```
2.
```

Man erkennt, dass einstellige Operationen nach dem Inputwert geschrieben werden (postfix). Dadurch sparen wir gegenüber der bekannten Schreibweise sqrt(sqrt(16)) 4 Klammern.

Programm 3.2: Berechne den Sinus von „pi halbe“.
```
pi:2 sin
```

Ergebnis:
```
1.
```

Programm 3.3: Berechne den Sinus von 30 Grad.
```
30:180*pi sin
```

Ergebnis:
```
0.5
```

Programm 3.4: Wie viele 10 stellige Dualzahlen gibt es?
```
2 hoch 10
```

Ergebnis:
```
1024
```

Programm 3.5: Berechne die Kantenlänge eines Würfels des Volumens 2 (die dritte Wurzel aus 2).
```
2 hoch 1/3
```

Ergebnis:
```
1.25992104989
```

oder
```
2 hoch (1:3)
```

Ergebnis:
```
1.25992104989
```

1/3 ist eine rationale Zahl. D.h., „/“ ist keine Operation im Gegensatz zu „:“. Da wir stets von links nach rechts rechnen, ergäbe 2 hoch 1 : 3 = (2 hoch 1):3 = 0.666666666667.

Programm 3.6: Bilde einen Durchschnitt.
```
1 3 2 1 3 4 ++:
```

Ergebnis:
```
2.33333333333
```

Um Schreibaufwand zu sparen, kann man bei einer Liste von Werten (hier die Zahlen der Zeile) auf jegliches sichtbare Trennzeichen und jegliche Klammerung verzichten. Auf diese Liste wird jetzt die Operation der Durchschnittsbildung `(++:)` angewandt. Man kann anstelle des Durchschnittes auch eine Summe (`++`), das Produkt (`**`), die Anzahl (`++1`), das Maximum (`max`), den Sinus (`sin`), `ln`, … bilden. Da `sin` nur einen Inputwert benötigt, ergibt sich bei Anwendung der Sinusoperation anstelle der Operation `++:` eine Liste von output-Werten. Auf diese Liste könnte man dann wieder `++:` anwenden und erhält dann lediglich eine Zahl. Obige Schreibweise ist zunächst vielleicht etwas gewöhnungsbedürftig, aber sie ist kompakter als die alte Schreibweise
```
avg([1; 3; 2; 1; 3; 4])
```
Insbesondere wenn mehrere Operationen hintereinander angewandt werden, spart man hierbei viele Klammern und vermindert damit Fehlerursachen. Wir betrachten `++:` und die anderen hier genannten Operationen als einstellig (unär), genau wie `sin` oder `sqrt` und schreiben sie nach dem Inputwert, da wir die gegebenen Zahlen als **eine** Liste (**ein** Tabment) ansehen.

Programm 3.7: Berechne den Wert des Terms
„`sqrt(abs(sin(7.1))+abs(cos(8.1)))`“.
```
7.1 sin abs
+ 8.1 cos abs
sqrt
```

Ergebnis:
```
0.986160835697
```

oder einzeilig mit einem Klammerpaar
```
7.1 sin abs + (8.1 cos abs) sqrt
```

Ergebnis:
```
0.986160835697
```

In der dreizeiligen Lösung wird, wie in Programmiersprachen üblich, von oben nach unten gerechnet. Den Wert der zweiten Zeile kann man nicht berechnen, wenn man nicht das „+“-Zeichen weglässt. Daher werden einfach die Werte der ersten und zweiten Zeile addiert. Aus dem Gesamtergebnis der zweiten Zeile wird dann durch die dritte Zeile die Wurzel gezogen.

Programm 3.8: Bilde das Produkt mehrerer Zahlen.
```
3 5 2 2 **
```

Ergebnis:
```
60
```

In o++o werden mehrere Operationen nach dem sehr alten „WaldPrinzip“ geschrieben. Für den Baum gibt es ein Wort und BaumBaum heißt Wald. Obiges `**` ersetzt daher 3 Multiplikationszeichen. Aufgrund dieser Operation können wir auf die Fakultätsfunktion verzichten. `9!` kann in o++o durch `1 ..9**` ausgedrückt werden.

Programm 3.9: Multipliziere jede Dezimalzahl einer Liste mit einer weiteren Zahl.

```
2.40 2.70 7.90 * 1.19
```

Ergebnis:

```
2.856 3.213 9.401
```

Hierbei wird jede Zahl der Inputliste mit 1.19 multipliziert. Sind die gegebenen Zahlen Nettopreise, so stellt das Ergebnis die zugehörigen Bruttopreise dar. Sind die gegebenen Zahlen dagegen Beträge einer Währung und ist 1.19 der Umrechnungskurs, so stellt das Ergebnis die Werte in der anderen Währung dar. Sind die Zahlen der Liste Längen von Rechtecken, so ergeben sich 3 Flächen von Rechtecken.
Wir sehen, dass Dezimalzahlen nicht mit dem Komma, sondern wie international üblich mit dem Punkt dargestellt werden.

Programm 3.10: Bilde die Summe von vielen positiven und vielen negativen Zahlen ohne viele Minuszeichen und Klammern zu benutzen.

```
4 5 3 2 1 8 9 ++
- 7 6 5 4 3 2 1 ++
```

Ergebnis:

```
4
```

Programm 3.11: Berechne die Umfänge mehrerer Kreise, wobei die Radien gegeben sind. Die Ergebnisse sind auf 2 Stellen nach dem Punkt zu runden.

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

Ergebnis:

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

Das Programm kann auch einzeilig geschrieben werden.

Programm 3.12: Berechne die Umfänge und Flächen mehrerer Kreise, wobei die Radien gegeben sind.

```
Rl:= 4 5 6 2 3.7 9.77
UMFANG:=R*pi*2
FLAECHE:=R*R*pi
rnd 1
```

Ergebnis:

```
R,  UMFANG, FLAECHE  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.
```

```
9.8 61.4    299.9
```

Durch das R bekommt jedes Element der gegebenen Liste den Namen (tag) R. Durch eine Zuweisung (:=) wird die gegebene Tabelle um eine neue Spalte erweitert. Oben kommen nacheinander die Spalten UMFANG und FLAECHE hinzu, so dass sich eine Tabelle vom Typ R,UMFANG,FLAECHE l ergibt. l steht für Liste. Diese kann bedauerlicherweise leicht mit der Eins (1) verwechselt werden.

Programm 3.13: Berechne die Flächen und Umfänge mehrerer Rechtecke.

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

Ergebnis:

```
A,   B,   UMFANG, FLAECHE  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
```

Die TAB-Klammern (<TAB!, !TAB>) sind nur im Programmteil des Systems erforderlich. Ist ein Tabment in einer Datei gespeichert, so erkennt das System das Format an der Endung (z.B. `.tab`). In der TAB-Darstellung müssen die Werte unter den zugehörigen Spaltennamen beginnen.

Programm 3.14: Bestimme den Gesamtpreis einer einfachen Rechnung.

```
<TAB!
ARTIKEL,  PREIS l
Bier      0.61
Brause    0.23
Schnitzel 2.40
!TAB>
++
```

Ergebnis:

```
3.24
```

Hier wird einfach die Summe über alle Zahlen der gegebenen Tabelle (Liste (l) von Paaren) gebildet. Die Artikelwerte sind Wörter und haben damit keinen Einfluss auf das Ergebnis. Ersetzt man ++ durch

```
*1.1
```

, so entsteht wieder eine Tabelle mit 2 Spalten und drei Zeilen (Sätzen, Tupeln), wobei jede Zahl ein Trinkgeld von 10% einschließt:

```
ARTIKEL,  PREIS  l
Bier      0.671
Brause    0.253
Schnitzel 2.64
```

Anschließend kann man wieder

```
++
```

anfügen, um auf die Gesamtsumme (hier `3.564`) zu kommen.

Programm 3.15: Bestimme den Gesamtpreis einer etwas komplizierteren Rechnung, die durch eine einfache Tabelle gegeben ist.

```
<TAB!
ARTIKEL,  PREIS, ANZAHL l
Bier      0.61   7
Brause    0.23   3
Schnitzel 2.40   4
!TAB>
POSPREIS:=PREIS*ANZAHL
++ POSPREIS
```

Ergebnis:

```
14.56
```

Durch die Zuweisung wird die gegebene Tabelle um eine neue Spalte mit dem Spaltennamen POSPREIS erweitert, wobei jeder der drei Preise mit der zugehörigen Anzahl multipliziert wird. Um die Gesamtsumme des Kneipenbesuchs zu ermitteln, muss man der `++`-Operation noch einen zweiten Inputwert geben. Ansonsten würde die Gesamtsumme aller 9 Zahlen der Zwischentabelle gebildet werden. Beide Programmzeilen können auch einfach durch

```
++ PREIS*ANZAHL
```

ersetzt werden. Der erste Inputwert einer Operation, die am Anfang einer Programmzeile steht, ist immer das Ergebnis der vorangehenden Programmzeile.
Das Ergibtzeichen `:=` der Zuweisung ist vom Gleichheitszeichen `=` zu unterscheiden. Das Gleichheitszeichen wie auch `<, >, <=`, in, … wird zur Formulierung von Bedingungen benötigt. Bedingungen dienen der Selektion -dem Streichen von (strukturierten) Zeilen.
Fügt man beispielsweise eine Bedingung

```
avec ARTIKEL=Bier
```

oder hier einfach

```
avec Bier
```

an, so ergibt sich im Endergebnis nur der Gesamtpreis für die 7 Bier. Will man den Preis für den Rest berechnen, so kann man stattdessen

```
sans ARTIKEL=Bier
```

oder einfach

```
sans Bier
```

einfügen.
Spaltennamen (Metadaten) müssen stets groß geschrieben werden. Die Schlüsselworte (gib, sans, avec, …) müssen stets klein geschrieben werden. Schreibt man ein Wort der Primärdaten immer mit Groß- **und** Kleinbuchstaben, so wird das Programm leichter lesbar.

Programm 3.16: Bestimme den Gesamtpreis eines längeren Kneipenbesuchs.

```
<TABH!
ARTIKEL,  PREIS,ANZAHLl l
Bier      0.61  7 6 5
```

```
                3
  Brause    0.23  3 4
  Schnitzel 2.40  4 3 2
  !TABH>
  POSPREIS:=PREIS*ANZAHL
  ++ POSPREIS
```

Ergebnis:

```
  36.02
```

Hierbei haben wir es mit einer strukturierten Tabelle zu tun, die für jede Position mehrere Bestellungen enthält. Da wir die horizontale Version TABH von TAB gewählt haben, müssen wir die ANZAHL-Einträge nicht untereinander schreiben, so dass man den Kneipenbesuch kompakt darlegen kann. Betrachtet man dagegen nicht das Endergebnis sondern nur das Ergebnis der Anwendung der Zuweisung, so müssen die ANZAHL- und POSPREIS-Werte untereinander stehen, da die entstehende strukturierte Tabelle ansonsten zu unübersichtlich werden würde:

```
  ARTIKEL,  PREIS, (ANZAHL, POSPREIS )l
  Bier      0.61    7       4.27
                    6       3.66
                    5       3.05
                    3       1.83
  Brause    0.23    3       0.69
                    4       0.92
  Schnitzel 2.4     4       9.6
                    3       7.2
                    2       4.8
```

Will man erst die ANZAHL-Werte addieren und dann multiplizieren,

```
  POSPREIS:= ANZAHLl ++ *PREIS
```

, so ergibt sich wieder eine kompaktere Zwischentabelle:

```
  ARTIKEL,  PREIS, POSPREIS,ANZAHLl m
  Bier      0.61   12.81    7 6 5 3
  Brause    0.23    1.61    3 4
  Schnitzel 2.4    21.6     4 3 2
```

Beide Programmzeilen können auch wieder durch `++ PREIS*ANZAHL` ersetzt werden.

Man kann die betrachteten Tabellen jeweils unter einem Namen beispielsweise mit der Endung `.tab bzw. .tabh` abspeichern und dann diese wieder aufrufen. Dadurch können Tabellen oder Dokumente in mehreren Programmen genutzt werden.

Programm 3.16 könnte dann so aussehen:

```
  kneipenbesuch.tabh
  ++ PREIS*ANZAHL
```

Dieses Beispiel lässt sich auf viele Anwendungen übertragen. Beim Kegeln könnte man ebenfalls eine Tabelle kegeln.tabh mit Wiederholgruppe aufbauen: NAME,WURFl m.

Die Namen und Höhe der einzelnen Würfe sind nach Belieben einzugeben.

`m` kürzt Menge und `l` Liste ab. Für jeden Kegler kann man die Gesamtsumme dann durch eine Zuweisung ermitteln und anschließend die Daten abfallend sortieren. Will man im Ergebnis nur noch bestimmte Spalten und will man nach diesen noch sortieren, so benötigt man eine gib-Klausel.

Programm 3.17: Bestimme für jede Person das Gesamtergebnis des Kegelns und sortiere die Daten danach.

```
kegeln.tabh
GESAMT:=WURFl ++
gib GESAMT,NAME m-
```

Es geht auch noch etwas kürzer:

```
kegeln.tabh
gib GESAMT,NAME m-
    GESAMT:= WURF! ++
```

Hierbei ergibt sich der Bezug der Aggregation (hier ++) aus dem Kopf der gewünschten Tabelle. GESAMT ist eine Aggregation pro NAME. Bei Mengen (m, m-) wird stets nach den zuerst angegebenen Spaltennamen sortiert.

Programm 3.18: Berechne den gewichteten Durchschnitt für 3 Schüler und den Gesamtdurchschnitt.

```
<TABH!
NAME,  KLAl, NOTEl l
Ernst  1 2   1 2 3 1 3 1 1
Clara  1 1   3
Sophia 1 3   1
!TABH>
DUR:=KLAl ++: *0.6 +(NOTEl ++: *0.4)
GESAMT:=DURl ++:
rnd 2
```

Ergebnis:

```
GESAMT, (NAME,  DUR, KLAl, NOTEl  l)
1.66     Ernst  1.59 1 2   1 2 3 1 3 1 1
         Clara  1.8  1 1   3
         Sophia 1.6  1 3   1
```

Programm 3.19: Eine Flasche mit Kork kostet eine Mark und zehn. Die Flasche ist eine Mark teurer als der Korken. Wie viel kostet der Korken?

```
FLASCHEl:= 0 .. 110
KORK := FLASCHE - 100
avec KORK+FLASCHE = 110
Ergebnis:
FLASCHE, KORK  l
105      5
```

Durch die erste Zuweisung bekommt jede der Zahlen von 0 bis 110 den Tag FLASCHE. Das erkennt man am besten, wenn man sich die ment-Darstellung ansieht:

```
<TABM>
  <FLASCHE>0</FLASCHE>
  <FLASCHE>1</FLASCHE>
  <FLASCHE>2</FLASCHE>
  <FLASCHE>3</FLASCHE>
  . . .
```

```
  <FLASCHE>108</FLASCHE>
  <FLASCHE>109</FLASCHE>
  <FLASCHE>110</FLASCHE>
</TABM>
```

Hätte man fälschlicherweise die Zuweisung `FLASCHE:= 0 ..110` geschrieben, so käme der Tag FLASCHE nur einmal vor:

```
<FLASCHE>
0
1
2
3
. . .
108
109
110
</FLASCHE>
```

zweite Lösung:

```
FLASCHEl:= 0 ..110
KORK1 := FLASCHE - 100
KORK2 := 110 - FLASCHE
avec KORK1=KORK2
```

Ergebnis:

```
FLASCHE, KORK1, KORK2  l
105      5      5
```

Diese Lösung ist vom methodischen Standpunkt vorteilhaft, da man sich die ersten 3 Programmzeilen durch einen Klick auf den Bildbutton illustrieren kann. Man erkennt, dass es sich hier um 2 Geraden handelt, deren Schnittpunkt durch die Bedingung bestimmt wird.

Programm 3.20: Schreibe 20 mal ich liebe dich:

```
“Je vous aime.“ mal 20
```

Resultat:

```
TEXTl
"Je vous aime." "Je vous aime." "Je vous aime." "Je vous aime."
"Je vous aime." "Je vous aime." "Je vous aime." "Je vous aime."
"Je vous aime." "Je vous aime." "Je vous aime." "Je vous aime."
"Je vous aime." "Je vous aime." "Je vous aime." "Je vous aime."
"Je vous aime." "Je vous aime." "Je vous aime." "Je vous aime."
```

`mal` ist eine binäre Operation, die wie alle anderen binären Operationen infix (zwischen die Inputwerte) geschrieben werden.

4 Schemata und TTDs strukturierter Tabellen

Wir fügen an dieser Stelle dieses Kapitel ein, um den Unterschied zwischen einfachen (flachen) und strukturierten Tabellen klarzustellen.

Zunächst ist eine Tabelle in n-Spalten A1,A2,…,An gegliedert. Die Spaltennamen im o++o-Programm dürfen keine Kleinbuchstaben enthalten. Besitzen alle A1,A2, …, An einen elementaren Typ, so enthält eine A1,A2, …, An-Tabelle genau eine einfache Zeile, z.B.:

```
NAME,ORT,  GEHALT
Paul Oehna 1000
```

Wenn wir ein Kollektionssymbol z.B. l für Liste anhängen, kann die Tabelle 0, 1 oder mehr Zeilen enthalten. Beispiel:

```
NAME,   ORT,    GEHALT l
Paul    Oehna   1000
Sophia  Dallgow  900
Claudia Dallgow 2000
Clara   Oehna    900
```

tab-Repräsentation einer flachen Tabelle (Liste von Tripeln)

NAME	ORT	GEHALT
Paul	Oehna	1000
Sophia	Dallgow	900
Claudia	Dallgow	2000
Clara	Oehna	900

graphik-Repräsentation der flachen Tabelle

Ersetzen wir das l durch das Mengensymbol m, so wird die Tabelle sortiert dargestellt.

```
NAME,   ORT,    GEHALT m
Clara   Oehna    900
Claudia Dallgow 2000
Paul    Oehna   1000
Sophia  Dallgow  900
```

Wenn diese Tabelle nach Ort sortiert werden soll, vertauscht man lediglich die Spalten in einer entsprechenden Anweisung `gib ORT,NAME,GEHALT m`

```
ORT,    NAME,   GEHALT m
Dallgow Claudia 2000
Dallgow Sophia   900
Oehna   Clara    900
Oehna   Paul    1000
```

Jetzt erkennt man, dass die Tabelle etwas Redundanz enthält. Mit einer strukturierten Tabelle kann man diese beseitigen (`gib ORT,(NAME,GEHALT m)m`):

```
ORT,   (NAME,   GEHALT  m) m
Dallgow Claudia 2000
        Sophia   900
Oehna   Clara    900
        Paul    1000
    (strukturierte_tabelle.tab)
```

Diese Tabelle enthält 2 Strupel (strukturierte Tupel=strukturierte Sätze=strukturierte records=strukturierte Elemente), d.h., z.B., dass die Anzahl (`++1`) der (jetzt strukturierten) Elemente der Tabelle nicht mehr 4 sondern 2 ist. Die Anzahl der Personen bleibt natürlich 4).

Trotzdem können wir diese Tabelle mit o++o genauso verarbeiten wie die vorangehenden. Aus der Ausgangstabelle kann man mit einer gib-Anweisung auch 2 oder 3 Tabellen erzeugen:

```
ORTm,   NAMEm,  GEHALTm
Dallgow Clara    900
Oehna   Claudia 1000
        Paul    2000
        Sophia
    (drei_kollektionen.tab)
```

Diese Gesamttabelle enthält zwar alle elementaren Daten der Ausgangstabelle, die weiteren Informationen sind jedoch verloren gegangen. Dallgow und Clara befinden sich in dieser tab-Darstellung zwar in der gleichen Zeile, dennoch wohnt Clara nicht in Dallgow. Man muss sich das Schema daher stets ansehen. Genau genommen, haben wir es hier mit einem Tripel von 3 Mengen zu tun:
{Dallgow Oehna}, {Clara Claudia Paul Sophia}, {900 1000 2000}
Hier steht das Komma für „Paarbildung“ und das Leerzeichen trennt die Elemente der Mengen.
Das Schema von `strukturierte_tabelle.tab` lautet:

```
ORT,(NAME,GEHALT m)m.
```

In vielen Fällen sind diese Metadaten ausreichend. Man erkennt aber hieran beispielsweise noch nicht, wie man mit welchen Spaltenwerten rechnen kann. Das macht die TTD (Tabment Typ Definition) deutlich:

```
TABMENT! (ORT,(NAME,GEHALT m) m)
GEHALT! ZAHL
NAME ORT! WORT
```

`TABMENT` ist ein Kunstwort aus TABelle und dokuMENT. Wird die Tabelle unter obigen Namen abgespeichert, so wird die TTD entsprechend erweitert:

```
TABMENT!  STRUKTURIERTE_TABELLE
STRUKTURIERTE_TABELLE! (ORT,(NAME,GEHALT m) m)
GEHALT! ZAHL
NAME ORT! WORT
```

Die Schreibweise der Kollektionssymbole ist wahrscheinlich etwas gewöhnungsbedürftig. Ein Kollektionssymbol wird wie die anderen einstelligen Operationen nach dem Schema geschrieben. Wie gesagt, das Schema ORT steht für einen Ort. D.h., eine o++o-Tabelle mit diesem Schema (mit diesem Kopf) kann nur einen Ort enthalten. Im Gegensatz dazu enthalten Relationen oder EXCEL-Tabellen mit dem Kopf ORT beliebig viele Orte. In o++o hat man drei Möglichkeiten, diese „Relationen“ darzustellen:

```
ORT            ORTl        ORTb       ORTm
Dallgow        Dallgow     Dallgow    Dallgow
Oehna          Oehna       Dallgow    Oehna
Dallgow        Dallgow     Oehna
Oehna          Oehna       Oehna
EXCEL       o++o-Liste   o++o-Bag   o++o-Menge
```

Bag ist englisch und heißt auch Multimenge. `mm` ist uns aber zu lang in der Schreibweise. Wir erkennen, dass Bags sortiert werden und Mengen sortiert werden und Mengen zusätzlich auch keine

Duplikate enthalten. Die Schemen `ORTm` und `ORT  m` drücken das gleiche aus. Der Unterschied wird erst bei mehreren Spalten sichtbar:

```
ORT,NAME m
```

enthält mehrere (`ORT,NAME`)-Paare, wogegen

```
ORT,NAMEm
```

nur einen Ort aber mit mehreren Namen enthalten kann. D.h., durch ein Kollektionssymbol, was einem Spaltennamen ohne Leerzeichen folgt, spart man ein Klammernpaar.

```
ORT,NAMEm
```

ist gleichwertig zu

```
ORT,(NAME m)
```

. Tabellen des Typs

```
ORT,NAMEm m
```

können dann viele verschiedene Tabellen des Typs

```
ORT,NAMEm
```

enthalten. Im Rahmen der `gib`-Anweisung enthält das Ergebnis von

```
gib ORT,NAMEm m
```

jeden Ort nur einmal. Damit kann

```
ORT,    NAMEm m
Dallgow Claudia
Dallgow Sophia
Oehna   Ernst
        Clara
```

nicht Ergebnis dieser gib-Anweisung sein, sondern:

```
ORT,    NAMEm m
Dallgow Claudia
        Sophia
Oehna   Ernst
        Clara
```

Der Vorteil dieser Strukturierung kommt erst richtig zum Tragen, wenn man zu jedem Ort noch weitere Daten speichert wie Einwohnerzahl, Bürgermeister,… . das innere m besagt, dass zu jedem Ort eine Menge von Namen existiert. Das heißt wieder, dass kein Name innerhalb eines Ortes mehrfach auftreten kann. Trotzdem könnte natürlich auch ein zweiter Ernst in Dallgow ausgegeben werden.
In o++o besitzt jedes Tabment ein Schema. Das trifft sogar für einzelne Werte zu, denen der Nutzer kein Schema zugeordnet hat. `Gerwisch` besitzt beispielsweise das elementare Schema `WORT` und 39175 das elementare Schema `ZAHL`. Damit ist Gerwisch ein Tabment. Gerwisch ist aber keine Tabelle des Relationalen Datenmodells im Gegensatz zu „{Gerwisch}“. Das scheint spitzfindig zu sein, hat aber großen Einfluss auf die Architektur des Datenmodells.

5 Einfache rekursive Zuweisungen

Neben den einfachen Berechnungen (:=) mit einer zugehörigen Formel, durch die eine Tabelle um eine neue Spalte erweitert wird, gibt es noch die sogenannten rekursiven Zuweisungen. Hierbei sind zwei Formeln gegeben. Die erste Formel ist für das erste Element der neuen Spalte der

entsprechenden Kollektion bestimmt und die zweite für die restlichen Elemente. Die Formel für die restlichen Elemente kann sich auf den Vorgänger der Spalte beziehen. Das ist im folgenden Programm `BETRAG pred`.

Programm 5.1: Was wird aus 100 Euro nach 10 Jahren bei 9 % Wachstum (Zinsen).

```
JAHRl:= 0 .. 10
BETRAG:= first 100. next BETRAG pred +% 9 at JAHR
rnd 2
```

Ergebnis:

```
JAHR, BETRAG  l
 0    100.
 1    109.
 2    118.81
 3    129.5
 4    141.16
 5    153.86
 6    167.71
 7    182.8
 8    199.26
 9    217.19
10    236.74
```

Programm 5.2: Wandle eine Dualzahl (hier 10111=23) in eine Dezimalzahl um.

```
BITl:= 1 0 1 1 1
DEZI:= first BIT next DEZI pred*2+BIT at BIT
```

Ergebnis:

```
BIT, DEZI  l
1     1
0     2
1     5
1    11
1    23
```

Ist nicht die ganze Entwicklung sondern nur das letzte Element der Liste gewünscht, so kann man noch die Bedingung `avec BIT pos- =1` oder einfach `last` anhängen. Ist das letzte Bit auch nicht gewünscht, so sollte `gib DEZI` folgen.

Programm 5.3. Wandle eine Dezimalzahl (hier 23) in eine Dualzahl um.

```
DURCHRESTl := first 23 divrest 2
              next  DURCHREST pred nth 1 divrest 2
              while DURCHREST ++ > 0
gib RESTl-
```

Ergebnis:

```
1 0 1 1 1
```

Hierbei werden mit einer rekursiven Erweiterung gleich 2 neue Spalten unter einem Namen eingeführt, DURCH und REST. `divrest` ist hier die ganzzahlige Division mit dem ganzzahligen Rest (ein Paar von Zahlen mit Tags DIV und REST).

Programm 5.4: Berechne die monatliche Entwicklung eines Hausbaukredits von 110 000 Euro bei 2% Jahreszins, wenn monatlich 900 Euro abgezahlt werden.

```
# Anfangskredit 110000 Euro
MONLAST:=900           # monatliche Belastung
PROZ:=1.02 hoch 1/12 -1 # 2 % Zinsen pro Jahr
KREDIT,ZINS,TILGUNG l:=
     first 110000.                  , 0.          , 0.
     next  KREDIT pred-(TILGUNG pred),(KREDIT*PROZ),(MONLAST-ZINS,KREDIT min)
     while KREDIT > 0 at PROZ
JAHR,MON:=KREDIT pos -1 divrest 12+1 leftat KREDIT
PROZ::=PROZ*100
rnd 2
```

Ergebnis :

```
MONLAST, PROZ, (JAHR, MON, KREDIT,   ZINS,  TILGUNG  l)

900      0.17    1     1   110000.     0.     0.
                 1     2   110000.   181.67 718.33
                 1     3   109281.67 180.49 719.51
                 1     4   108562.16 179.3  720.7
                 1     5   107841.46 178.11 721.89
                 1     6   107119.57 176.92 723.08
                 1     7   106396.49 175.72 724.28
                 1     8   105672.21 174.53 725.47
                 1     9   104946.74 173.33 726.67
                 1    10   104220.06 172.13 727.87
                 1    11   103492.19 170.93 729.07
                 1    12   102763.12 169.72 730.28
                 2     1   102032.84 168.52 731.48
                 2     2   101301.35 167.31 732.69
                 2     3   100568.66 166.1  733.9
                 2     4    99834.76 164.89 735.11
                 . . .
                11    12     5930.53   9.79 890.21
                12     1     5040.33   8.32 891.68
                12     2     4148.65   6.85 893.15
                12     3     3255.5    5.38 894.62
                12     4     2360.88   3.9  896.1
                12     5     1464.78   2.42 897.58
                12     6      567.2    0.94 567.2
```

Etwas anspruchsvollere rekursive Zuweisungen (mit o++o-Nummer) findet man im Kapitel 8.

6 Anfragen

Wir nehmen jetzt an, dass jeder Fluss in `fluesse.tabh` eine `LAENGE`-Spalte hat. Dann findet man alle Flüsse, die länger als 500 km sind durch folgende Anfrage:

Programm 6.1: Selektiere in einer vorgegebenen Tabelle.

```
fluesse.tabh
avec LAENGE > 500
```

„avec“ ist französisch und heißt „mit“. Analog benutzen wir für „ohne“ „sans“.
Jetzt sollen die Flüsse noch nach der Länge sortiert werden und mit Nebenflüssen ausgegeben werden:

Programm 6.2: Selektion mit anschließender Sortierung.

```
aus  fluesse.tabh
avec LAENGE>520
gib  LAENGE,FLUSS,NEBENFLUSSm m
```

Ergebnis:

```
LAENGE, FLUSS, NEBENFLUSSm   m
 544    Main   "Fraenkische Saale" Gersprenz Kinzig
               Lohrbach Nidda Rechtenbach Regnitz
               "Roter Main" Tauber "Weisser Main"
 544    Mosel  Alf Dhron Elzbach Kyll Lieser Madon Meurthe
               Moselotte Orne Ruwer Saar Salm Sauer Seille
               Vologne
 544    Saar   Blies Nied Prims Rossel
 866    Oder   Bartsch Birawka Bober Eilang Faule Obra
               Glatzer Neisse Hotzenplotz Ihna Iseritz
               Kaltebach Katzbach Klodnitz Lohe Malapane
               Mietzel Neisse Ohle Olsa Oppa Ostrawitza
               Pleiske Raude Roehrike Stober Summina Thue
               Tiefenburgbach Warthe Weide Weistritz
               Weissfurth Welse Zinna
1165    Elbe   Adler Aland Alster Biela Bille Dahle
               Doellnitz Eger Elde Este Gottleuba Havel
               Ilmenau Iser Jahna Jeetze Kamnitz Kirnitzsch
               Lachsbach Lockwitz Loecknitz Model Moldau
               Mulde Mueglitz Ohre Oste Polzen Priessnitz
               Saale "Schwarze Elster" Stepenitz Stoer
               Tanger Weisseritz Wesenitz "Wilde Sau"
               Zidlina
1320    Rhein  Aare Ahr Alb Alb bei Albbruck Argen Birs
               "Bregenzer Ache" Duessel Elde Emscher Erft
               Glatt Hinterrhein Ill Kander Kinzig
               Kraichbach Lahn Lauter Leiblach Leimbach
               Leopoldskanal Lippe Main Mosel Murg Nahe
               Neckar Queich Ruhr Schussen Sieg Speyerbach
               Thur Toess Vorderrhein Weschnitz Wied Wiese
```

```
                Wupper Wutach
  2845    Donau Abens Altmuehl Blau Breg Brenz Brigach Cerna
                Donaudelta Donaukanal Drau Eipel Enns Fischa
                Friedberger Ach Grosse Laaber Grosse Muehl
                Guenz Hron Iller Ilz Inn Ipel Isar Jantra
                Jiu Kamp Kleine Paar Krems Lauchert Lech
                Leitha March Mindel Morava Naab Olt Paar
                Pruth Raab Regen Riss Save Schmutter
                Schwechat Temesch Theiss Timok Traisen Traun
                Vils Vah Woernitz Ybbs Zusam
```

Will man die längsten Flüsse zuerst ausgeben, muss man lediglich, das äußere `m` durch `m-` ersetzen.

Unsere Datei `fluesse.tabh` enthält für jeden Fluss auch eine weitere Wiederholgruppe `LAND,BUNDESLANDm m`, die angibt durch welche Länder und Bundesländer der Fluss fließt. In `fluesse.tabh` ist also der FLUSS dem Bundesland übergeordnet. Will man das umkehren, kann das wieder einfach durch Angabe des Kopfes der gewünschten Tabelle realisiert werden.

Programm 6.3: Umstrukturierung der Fluesse-Datei.

```
aus fluesse.tabh
gib BUNDESLAND,FLUSSm m
```

Ergebnis:

```
BUNDESLAND,             FLUSSm  m
Baden Württemberg       Donau Iller Neckar Rhein
Bayern                  Abens Donau Guenz Iller Inn Isar
                        Lech Main Mindel Naab Regen Saale
                        Wertach Woernitz
Berlin                  Havel Spree
Brandenburg             Aland Elde Havel Neisse Oder Spree
                        Stepenitz Uecker
Bremen                  Weser
Hamburg                 Alster Elbe Este Wandse
Hessen                  Fulda Main Neckar Rhein Werra Weser
Mecklenburg Vorpommern  Aland Elde Havel Loecknitz Oder
                        Peene Stepenitz Uecker Warnow
Niedersachsen           Elbe Este Fulda Jeetze Loecknitz
                        Oste Werra Weser
NordrheinWestfalen      Rhein Ruhr Weser Wupper
RheinlandPfalz          Mosel Rhein Saar
Saarland                Saar
Sachsen                 Elbe Neisse Spree
Sachsen Anhalt          Bode Elbe Havel Ilm Saale Unstrut
Schleswig Holstein      Alster Elbe Stoer Trave Wandse
Thüringen               Ilm Saale Unstrut Werra
```

Durch diese freie Umstrukturierung werden zu jedem Bundesland alle Flüsse, die durch dieses fließen, gesammelt. Die Bundesländer und die Flüsse innerhalb der Bundesländer werden sortiert. Ein Fluss kann hier in mehreren Bundesländern vorkommen. Jedes Bundesland erscheint nur einmal, da wir als äußere Kollektion eine Menge gewählt haben. Den Beginn eines Programmteils kann man auch mit `aus` kennzeichnen.

Programm 6.4: Bilde eine Vereinigung (alle StudentenIDs, die in einer der Tabellen enthalten sind).

```
aus examen.tab, projekte.tab
gib STIDm
```

Resultat (tabh):

```
STIDm
1234 1245 3456 4567 5678
```

Hierbei seien examen.tab und projekte.tab folgende Tabellen:

```
STID, KURS,       NOTE  m,(STID, PROJ, STUNDEN  m)
1234  Algebra     1        1234  Fritz  4
1234  Geschichte  1        1234  Otto   2
1234  Logik       2        1245  Konfuz 5
1245  Algebra     3        1245  Ming   4
1245  Datenbanken 1        1245  Otto   6
1245  Otto        1        4567  Monet 10
3456  Datenbanken 1        5678  Monet 20
3456  OCaml       2
5678  Apel        1
5678  Repin       1
```

Programm 6.5: Bilde einen Durchschnitt (alle StudentenIDs, die in beiden Tabellen enthalten sind).

```
aus examen.tab, projekte.tab
igib STIDm
```

Resultat:

```
STIDm
1234 1245 5678
```

Mit *igib* (siehe Abschnitt 7) können auch „joins“ ausgedrückt werden und damit auch Anfragen an ganze Datenbanken formuliert werden. Die folgenden Anfragen beziehen sich auf eine Datenbank `uni.tab`, deren zweite Tabelle unstrukturiert ist. Die Anfragen müssen nicht oder nur schwach modifiziert werden, wenn alle Tabellen von `uni.tab` unstrukturiert (relational) sind.

```
FAK,  DEKAN,  BUDGET, STUDKAP m
Inf   Reichel 10000   500
Kunst Sitte    2000   600
Mathe Dassow   1000   200
Philo Hegel    1000    10


STID, NAME,  ORT?,    STIP, FAK,  (KURS,       NOTE m),(PROJ, STUNDEN m)
m
1234  Ernst  Oehna    500   Mathe  Algebra     1        Fritz  4
                                  Geschichte  1        Otto   2
                                  Logik       2
1245  Sophia Berlin   400   Inf    Algebra     3        Konfuz 5
                                  Datenbanken 1        Ming   4
                                  Otto        1        Otto   6
```

```
3456  Clara  Oehna    450   Inf    Datenbanken 1
                                   OCaml       2
4567  Ulrike          400   Kunst                      Monet 10
5678  Kaethe Gerwisch   0   Kunst  Apel        1       Monet 20
                                   Repin       1
```

Die Datenbank besteht aus 2 Tabellen. Die Tabelle “STUDENTEN“ ist strukturiert, da zu jedem Satz (Satz=strukturiertes Tupel = Student) 7 Komponenten gespeichert werden und die letzten beiden Komponenten selbst wieder kleine Tabellen sind. Die vorletzte Komponente von Clara enthält beispielsweise zwei Prüfungsergebnisse und die letzte die leere Menge von Projekten. Die Tabelle “FAKS“ ist relational.

Programm 6.6: Berechne das Gesamtbudget der Uni.

```
aus uni.tab
++  BUDGET
```

Resultat:

```
14000
```

Programm 6.7: Berechne das Gesamtbudget und das Durchschnittsbudget der Uni.

```
aus uni.tab
gib SUMBUD,DURBUD
    SUMBUD:= BUDGET! ++
    DURBUD:= BUDGET! ++:
```

Resultat:

```
SUMBUD, DURBUD
14000   3500.
```

Da die letzten beiden Zeilen mit mehr als 3 Leerzeichen beginnen, gehören sie vom logischen Standpunkt noch zur vorangehenden Zeile.

Programm 6.8: Wie viele Fakultäten und Studenten hat die Uni.

```
uni.tab
++1
```

Resultat:

```
4     5
```

Programm 6.9: Gib zu jedem Kurs die zugehörigen Studenten mit Note.

```
aus uni.tab
gib KURS,(NAME,NOTE b)m
```

Resultat:

```
KURS,       (NAME,  NOTE  b) m
Algebra      Ernst  1
             Sophia 3
Apel         Kaethe 1
Datenbanken Clara  1
             Sophia 1
```

```
Geschichte  Ernst  1
Logik       Ernst  2
OCaml       Clara  2
Otto        Sophia 1
Repin       Kaethe 1
```

Programm 6.10: Gib alle Sätze (Strupel), die 1234 enthalten.

```
aus  uni.tab
avec 1234
```

Resultat als hsq-(hierarchisch sequentiell)Ausgabe:

```
STID NAME ORT STIP FAK
 KURS NOTE
  PROJ STUNDEN
   FAK DEKAN BUDGET STUDKAP
1234 Ernst Oehna 500 Mathe
 Algebra 1
 Geschichte 1
 Logik 2
  Fritz 4
  Otto 2
```

Programm 6.11: Selektiere in allen Tabellen, die den Studentenidentifikator enthalten, nach dem Studenten mit der Nummer 1234.

```
uni.tab
avec STID=1234
```

Resultat:

```
STID NAME ORT STIP FAK
 KURS NOTE
  PROJ STUNDEN
   FAK DEKAN BUDGET STUDKAP
1234 Ernst Oehna 500 Mathe
 Algebra 1
 Geschichte 1
 Logik 2
  Fritz 4
  Otto 2
   Inf Reichel 10000 500
   Kunst Sitte 2000 600
   Mathe Dassow 1000 200
   Philo Hegel 1000 10
```

Programm 6.11 unterscheidet sich nur bzgl. der FAKS-Tabelle von Programm 6.10. Im Ergebnis von Programm 6.10 ist diese leer und in Programm 6.11 bleibt sie vollständig erhalten. In anderen Anwendungen könnte man statt des Studentenidentifikators (STID) auch Artikelnummern oder Teilenummern, … wählen.

7 Anfragen an mehrere Tabellen ohne Joins (igib)

Programm 7.1: Gib zum Studenten mit der Nummer 1234 den Dekan und seine Examen.

```
aus  uni.tab
avec STID=1234
igib NAME,DEKAN,(KURS,NOTE m)m
```

Resultat:

```
NAME, DEKAN, (KURS,      NOTE  m) m
Ernst Dassow  Algebra    1
              Geschichte 1
              Logik      2
```

Hier wird ein „Join“ ohne Joinbedingung realisiert. Würde man anstelle von i`gib gib` verwenden, so würde die Ergebnismenge leer bleiben, da `gib` keine Verbindung zwischen NAME und DEKAN über FAK herstellt.

Programm 7.2: Berechne die Anzahl der Einsen für die einzelnen Fakultäten.

```
aus  uni.tab
avec NOTE=1
gib  FAK,ANZ m
     ANZ:=NOTE! ++1
```

Resultat:

```
FAK,  ANZ  m
Inf   3
Kunst 2
Mathe 2
Philo 0
```

Programm 7.3: Teste das folgende Programm.

```
aus  uni.tab
avec DEKAN in [Reichel Dassow]
gib  FAK,DEKAN,(NAME,STIP m)m
```

Resultat:

```
FAK,  DEKAN, (NAME, STIP  m) m
Inf   Reichel
Mathe Dassow
```

Programm 7.4: Gib zu den Fakultäten von Dassow und Reichel alle zugehörigen Studenten.

```
aus  uni.tab
avec DEKAN in [Reichel Dassow]
igib FAK,DEKAN,(NAME,STIP m)m
```

Resultat:

```
FAK,  DEKAN, (NAME,  STIP  m) m
Inf   Reichel Clara  450
              Sophia 400
Mathe Dassow  Ernst  500
```

Programm 7.5: Gesucht sind alle Fakultäten, die Studenten mit mindestens zwei Einsen und einer Drei haben.

```
aus  uni.tab
avec {{1 1 3}} inmath NOTEb
igib FAK,DEKAN m
```

Ergebnis:

```
FAK, DEKAN  m
Inf  Reichel
```

„b“ steht für bag (Multimenge).

Programm 7.6: Gesucht sind alle Studenten, die genau 2 Einsen haben.

```
aus  uni.tab
avec NOTEl = [1 1]
gib  STUDENTEN
```

Resultat (hsq):

```
STID NAME ORT STIP FAK
 KURS NOTE
  PROJ STUNDEN
5678 Kaethe Gerwisch 0 Kunst
 Apel 1
 Repin 1
  Monet 20
```

Programm 7.7: Gesucht sind alle Daten von Ernst (einschließlich seiner Fakultätsdaten).

```
aus  uni.tab
avec NAME=Ernst
igib
```

Resultat:

```
STID NAME ORT STIP FAK
 KURS NOTE
  PROJ STUNDEN
   FAK DEKAN BUDGET STUDKAP
1234 Ernst Oehna 500 Mathe
 Algebra 1
 Geschichte 1
 Logik 2
  Fritz 4
  Otto 2
   Mathe Dassow 1000 200
```

Das Programm muss nicht verändert werden, wenn alle Tabellen von `uni.tab` in erster Normalform (unstrukturiert) sind.

Programm 7.8: Bestimme den Gesamtpreis eines längeren Kneipenbesuchs, wobei eine separate Preistabelle gegeben ist.

```
<TABH!
ARTIKEL,  ANZAHLl m
```

```
Bier      2 4 5
Brause    3 2
Schnitzel 4 3
!TABH>
, artikels.tab
igib TOT,(ARTIKEL,TOT m)  TOT:=ANZAHL*PREIS! ++
```

Ergebnis:

```
TOT,  (ARTIKEL,  TOT2  m)
79.3   Bier      29.7
       Brause    16.
       Schnitzel 33.6
```

```
artikels.tab:
ARTIKEL,  PREIS  m
Bier      2.7
Brause    3.2
Schnitzel 4.8
Steak     4.8
```

Programm 7.9: Gesucht ist der Dekan von Ernst. (Anfrage an eine relationale Datenbank.)

```
unirelational.tab
avec NAME=Ernst
igib DEKAN
```

Resultat:

```
DEKAN
Dassow
```

unirelational.tab besitzt hierbei das folgende Schema:

```
TABMENT! UNIRELATIONAL
UNIRELATIONAL! STUDENTEN,EXAMEN,PROJEKTE,FAKS
PROJEKTE! (STID,PROJ,STUNDEN m)
STUDENTEN! (STID,NAME,ORT?,STIP,FAK m)
EXAMEN! (STID,KURS,NOTE m)
FAKS! (FAK,DEKAN,BUDGET,STUDKAP m)
```

Programm 7.10: Gib mir alle Daten von Ernst.

```
unirelational.tab
avec NAME=Ernst
igib
```

Ergebnis als hsq-Ausgabe:

```
STID NAME ORT STIP FAK
 STID KURS NOTE
  STID PROJ STUNDEN
   FAK DEKAN BUDGET STUDKAP
1234 Ernst Oehna 500 Mathe
 1234 Algebra 1
```

```
 1234 Geschichte 1
 1234 Logik 2
  1234 Fritz 4
  1234 Otto 2
   Mathe Dassow 1000 200
```

Es sei angemerkt, dass alle Daten von Ernst im Ergebnis erscheinen, ohne dass ein Kreuzprodukt benutzt wird. Ansonsten würden die Projekte „Fritz“ und „Otto“ jeweils 3 mal erscheinen.

Programm 7.11: Gesucht sind die Prüfungsergebnisse vom Studenten mit der Nummer 1234.

```
unirelational.tab
avec STID=1234
igib NAME,DEKAN,(KURS,NOTE m)m
```

Ergebnis:

```
NAME, DEKAN, (KURS,      NOTE  m) m
Ernst Dassow  Algebra    1
              Geschichte 1
              Logik      2
```

Es sei angemerkt, dass das Ergebnis Informationen aus 3 Tabellen enthält, ohne dass eine Joinbedingung nötig ist.

8 Eine Stücklistenauflösung mit o++o-Nummer

Programm 8.1: Gesucht sind alle Baugruppen und Einzelteile des Polos und des Motors.

```
<TAB!
OT,         EIGENSCHAFT,(UT,         ANZ m) m
Buchse      zylindrisch
Felge       glatt
Polo        modern       Rad         4
                         Motor       1
                         Karosse     1
Golf        schnell      Rad         4
                         Reserverad  1
                         Karosse     1
                         Klimaanl    1
                         Motor       1
Karosse     blau
Klimaanl    robust
KolbRing    rund
Kolben      leicht       KolbRing    2
                         Buchse      1
Motor       schwer       Kolben      6
                         Schraube    8
Rad         rund         Schraube    5
                         Reifen      1
                         Felge       1
```

```
Reserverad rund         Schraube   4
                        Reifen     1
                        Felge      1
Reifen     schwarz
Schraube   stabil
!TAB>
onrs OTTONR ! [Motor Polo] # Einführung der o++o-Nummern (ONR)
RANZ:=firstonr ANZ nextonr RANZ pred * ANZ at ANZ # ONR-Rekursion
gib OT,(UT,TOT m) m    TOT:=RANZ!++
```

Zwischenergebnis nach der ersten Anweisung:

```
OT,   EIGENSCHAFT, (OTTONR, UT,      ANZ  m) l
Motor schwer        1       Schraube 8
                    2       Kolben   6
                    2.1     KolbRing 2
                    2.2     Buchse   1
Polo  modern        1       Rad      4
                    1.1     Schraube 5
                    1.2     Reifen   1
                    1.3     Felge    1
                    2       Motor    1
                    2.1     Schraube 8
                    2.2     Kolben   6
                    2.2.1   KolbRing 2
                    2.2.2   Buchse   1
                    3       Karosse  1
```

Zwischenergebnis nach Anwendung der rekursiven o++o-Nummer Zuweisung:

```
OT,   EIGENSCHAFT, (OTTONR, UT,      ANZ, RANZ  m) l
Motor schwer        1       Schraube 8     8
                    2       Kolben   6     6
                    2.1     KolbRing 2    12
                    2.2     Buchse   1     6
Polo  modern        1       Rad      4     4
                    1.1     Schraube 5    20
                    1.2     Reifen   1     4
                    1.3     Felge    1     4
                    2       Motor    1     1
                    2.1     Schraube 8     8
                    2.2     Kolben   6     6
                    2.2.1   KolbRing 2    12
                    2.2.2   Buchse   1     6
                    3       Karosse  1     1
```

Endergebnis:

```
OT,  (UT,      TOT  m) m
Motor Buchse    6
```

```
       Kolben    6
       KolbRing 12
       Schraube  8
Polo   Buchse    6
       Felge     4
       Karosse   1
       Kolben    6
       KolbRing 12
       Motor     1
       Rad       4
       Reifen    4
       Schraube 28
```

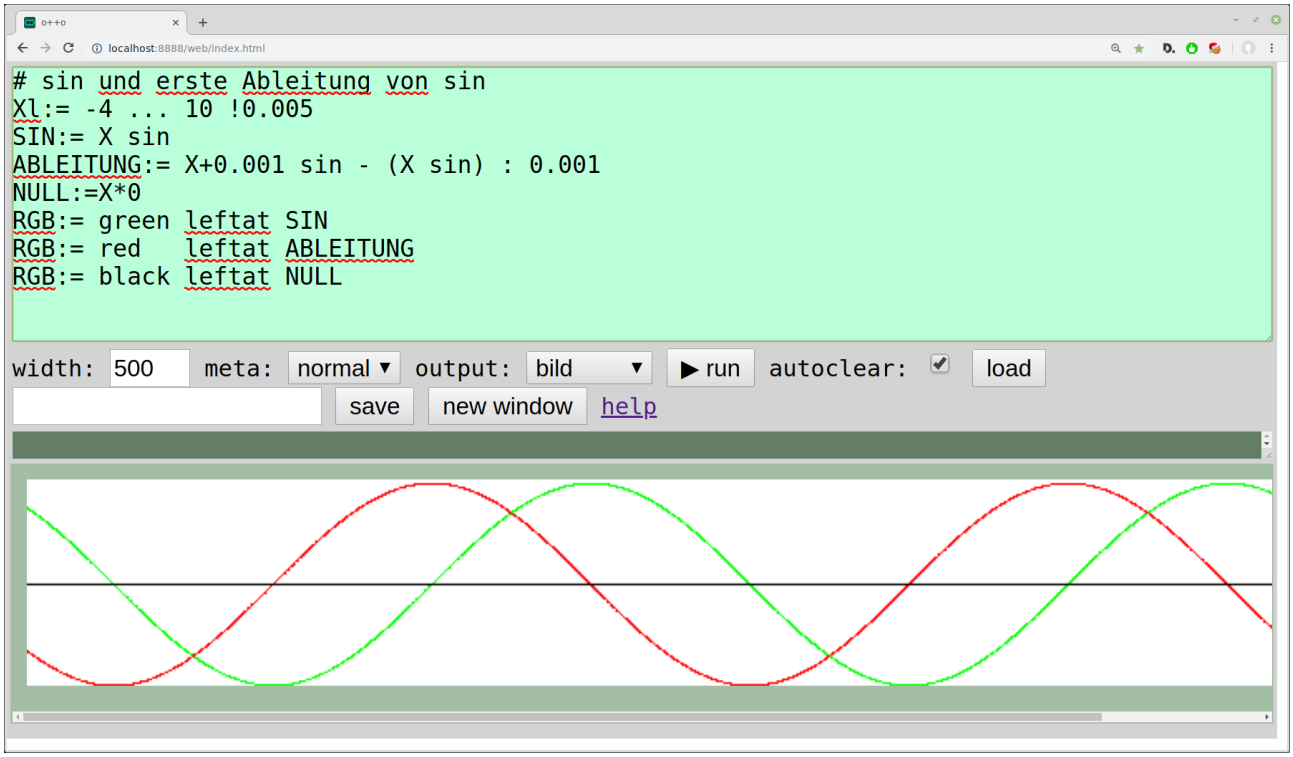
9 Bilder generieren

Programm 9.1: Gib die Punkte von drei Funktionen mehrfarbig aus.

```
# sin und erste Ableitung von sin
Xl:= -4 ... 10 !0.005
SIN:= X sin
ABLEITUNG:= X+0.001 sin - (X sin) : 0.001
NULL:=X*0
RGB:= green leftat SIN
RGB:= red   leftat ABLEITUNG
RGB:= black leftat NULL
```

Auszug aus der Ergebnistabelle (ohne Farbwerte) von 2800 Punkten:

```
X,                SIN,               ABLEITUNG,          NULL  l
-4.                0.756802495308    -0.654021913139     -0.
-3.995             0.75352483081     -0.657796099859     -0.
-3.99              0.75022832823     -0.661553841711     -0.
-3.985             0.746913069981    -0.665295044752     -0.
-3.98              0.743579138944    -0.66901961545      -0.
-3.975             0.740226618468    -0.672727460694     -0.
-3.97              0.736855592364    -0.676418487786     -0.
-3.965             0.73346614491     -0.680092604451     -0.
. . .
 9.96             -0.510032040244    -0.859900243999      0.
 9.965            -0.514326423954    -0.857337195738      0.
 9.97             -0.518607949529    -0.854752714092      0.
 9.975            -0.522876509934    -0.852146863673      0.
 9.98             -0.527131998452    -0.849519709627      0.
 9.985            -0.531374308698    -0.846871317632      0.
 9.99             -0.535603334614    -0.844201753898      0.
 9.995            -0.539818970475    -0.841511085165      0.
```



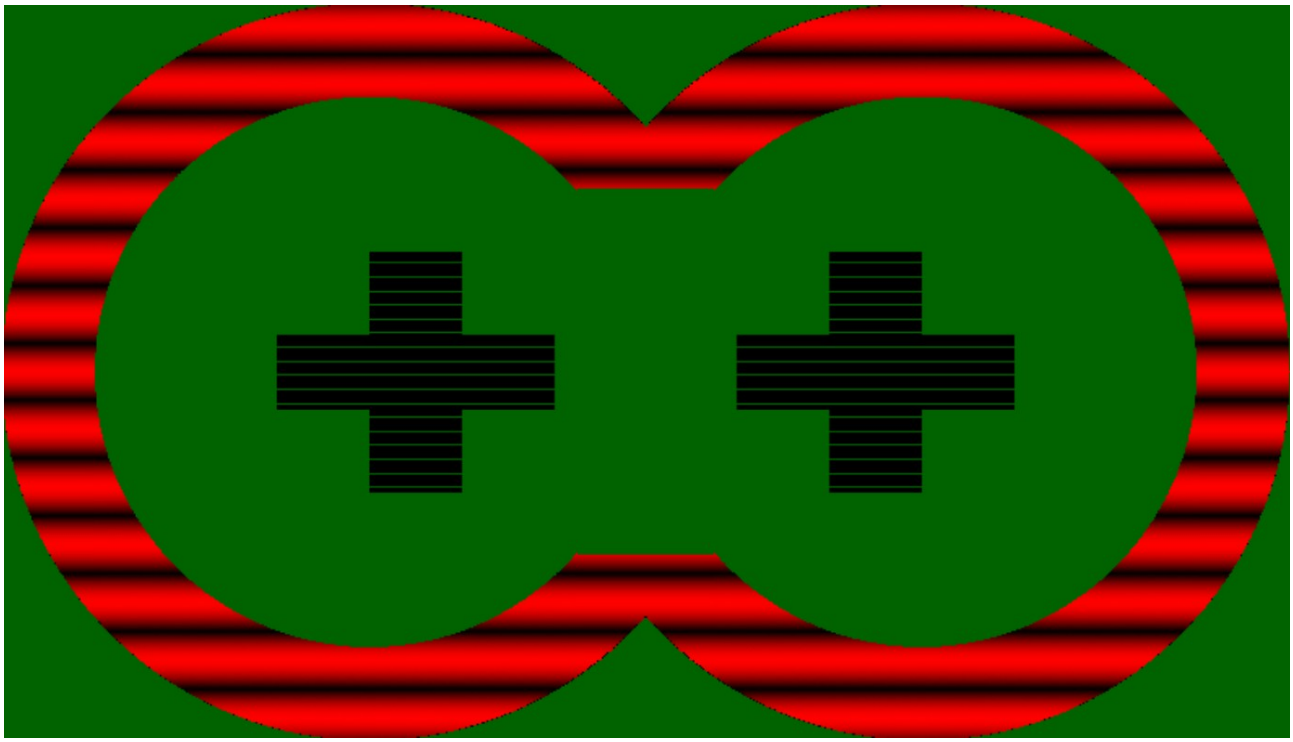
Bis auf die Farbwerte wird hier nichts anderes gemacht als für jedes der 2800 Elemente dreimal einen Punkt zu setzen. Dieses Verfahren – eine Wertetabelle aufzustellen- lernt jeder Schüler. Hier führt dieses einfache Methode bereits zu einer Visualisierung. Obwohl hier lediglich einzelne Punkte ausgegeben werden, hat man den Eindruck von Funktionsverläufen. Mit geeigneten o++o-Programmen können nicht nur einzelne Funktionen sondern auch ganze Bilder, wie Fahnen und Farbverläufe mit dem bild-Button erzeugt werden.

Programm 9.2: Generierung eines o++o Logos:

```
RGB:=darkgreen
X3l := -2 ... 5!0.02
Y3l:= -2 ... 2!0.02 at X3
=: $HINTERGRUND
aus empty_t
Xl:= -2. ... 1.5!0.02
Y1:= 4 - (X*X) sqrt
Y2:=if X< -1.5 then 0 else if X >1.13 then 1. else  2.25 - (X*X) sqrt
Zl:= Y2 ... Y1!0.02 at Y2
RGB:=10*Z cos abs,,1,,0 sin abs leftat Z
STUFE:=if -0.5 <X & X< 0. | (0.5<X & X<1.) then 0.2 else (if -0. <X & X<
0.5 then 0.65)
Wl := 0 ... STUFE!0.01 at STUFE
RGB:=black leftat W #1,,10*W*X sin abs,0  leftat W
=: $OBENLINKS
$OBENRECHTS:= $OBENLINKS *(-
1,1,1,1,1,1,1,1,1,1,1,1)+(3,0,0,0,0,0,0,0,0,0,0,0)
$UNTENLINKS:=$OBENLINKS * (1,-1,-1,-1,1,1,1,-1,1,1,1,-1)
$UNTENRECHTS:=$OBENRECHTS*(1,-1,-1,-1,1,1,1,-1,1,1,1,-1)
```

```
aus $HINTERGRUND,$OBENLINKS,$OBENRECHTS,$UNTENRECHTS,$UNTENLINKS
,<TAB!
X,  Y l
-30 -30
!TAB>
```

Das Logo kann wieder mit dem bild-output erzeugt werden:



Das Programm benötigt momentan relativ viel Zeit (mehr als eine Sekunde), deswegen sollte man eine solche Rechnung nur mit einem lokalen Server durchführen. Wenn man die Schrittweiten verfeinert, verschlechtern sich die Zeiten. Man erhält durch kleine Modifikationen im Programm vielfältige Veränderungen im Bild. Im Abschnitt 12 (o++o für die Schule) wird ein weiteres Beispiel (12.16) für die Generierung von Bildern aufgeführt.

10 Diagramme

Programm 10.1: Berechne den Durchschnitts-BMI pro Alter und den BMI pro Person und Alter für alle Personen über 20 sowie den Gesamtdurchschnitt.

```
<TAB!
NAME,      LAENGE,  (ALTER, GEWICHT l) l
Klaus      1.68      18     61
                     30     65
                     56     80
Rolf       1.78      40     72
Kathi      1.70      18     55
                     40     70
Walleri    1.00       3     16
```

```
Viktoria   1.61      13     51
Bert       1.72      18     66
                     30     70
!TAB>
avec NAME! 20<ALTER
gib BMI,(ALTER,BMI,(NAME,BMI m) m) BMI:=GEWICHT:LAENGE:LAENGE!++:
rnd 1
```

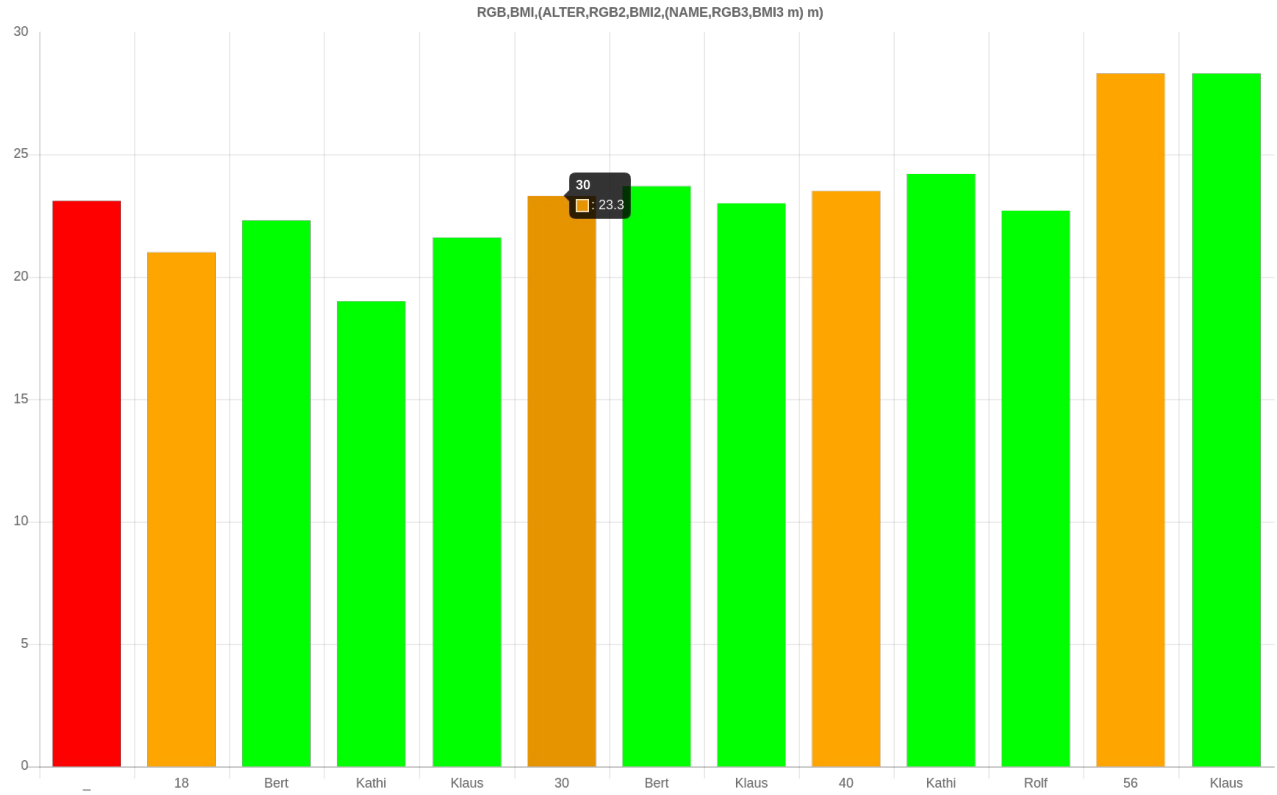
Die TAB-Klammern deuten an, dass die eingeschlossenen Daten der TAB-Darstellung entsprechen. Die obige Bedingung selektiert Personen-Sätze, d.h. `NAME,LAENGE,(ALTER,GEWICHT l)` Tupel (strukturierte Tupel bzw. Strupel). Da eine Person mehrere ALTERs-Angaben besitzt, muss quantifiziert werden. `NAME! 20<ALTER` selektiert demnach alle Personen, die einen entsprechenden Alterseintrag besitzen. D.h., der Existenzquantor wird nicht geschrieben, gehört aber zu jeder Bedingung. In diesem kleinen Beispiel könnte man die Selektion natürlich auch per Hand realisieren.

Resultat:

```
BMI, (ALTER, BMI2, (NAME, BMI3  m) m)
23.1  18     21.    Bert  22.3
                    Kathi 19.
                    Klaus 21.6
      30     23.3   Bert  23.7
                    Klaus 23.
      40     23.5   Kathi 24.2
                    Rolf  22.7
      56     28.3   Klaus 28.3
```

Das Beispiel zeigt, das man eine Hierarchie einfach durch Angabe des gewünschten Schemas umkehren kann. Im Ergebnis ist der Name dem Alter untergeordnet.
Das Ergebnis kann beispielsweise durch zwei Klicks als Säulendiagramm dargestellt werden. Dazu ordnen wir jeder Ebene eine bestimmte Farbe zu. Die letzte Zeile ist erforderlich, damit das Alter nicht als Säule sondern als Unterschrift dargestellt wird.

```
RGB:=red leftat BMI
RGB2:=orange at ALTER
RGB3:=green at NAME
ALTER::=ALTER wort
```

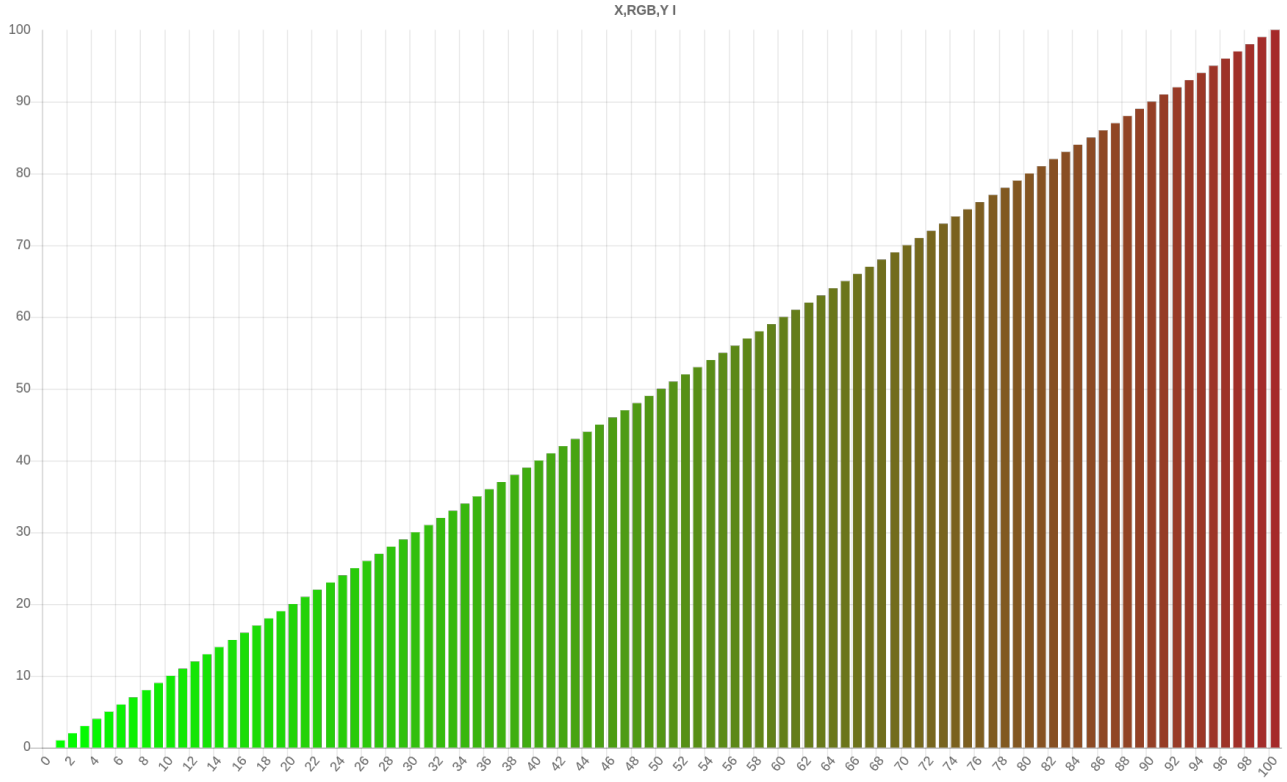


Hier erkennt man die Vorteile von Diagrammen, die auf strukturierten Tabellen basieren.

1. Die übergeordneten Säulen können hervorgehoben werden (nullte Ebene rot; übergeordnete erste Ebene orange, zweite Ebene grün)
2. Die Spaltenbenennungen sind kürzer und übersichtlicher, da sich die Werte der übergeordneten Säulen nicht wiederholen müssen (anstatt von Bert30 und Klaus30 muss beispielsweise lediglich Bert und Klaus ausgegeben werden)

Programm 10.2: Ein Diagramm mit Farbverlauf von grün nach braun.

```
Xl:= 0 ..100
Y:= X at X
RGB:= (green*(100-X) :100)+ (brown *X :100) leftat Y
X::=X wort
```



11 „Hallo otto“ - Spielerei

Programm 11.1: Gib zwei Worte aus.

```
Hallo otto
```

Das Ergebnis ist eine Liste von zwei Wörtern (WORTl).

Programm 11.2: Gib ein Paar von zwei Worten aus.

```
Hallo, otto
```

Das Ergebnis ist ein Paar von 2 Worten (WORT,WORT). Das Komma trennt Komponenten von Tupeln. Ein Tupel (das Paar ist ein 2-Tupel) wird in der Regel horizontal ausgegeben und Elemente von Kollektionen (Listen, Mengen, Multimengen) in der Regel vertikal. Um Platz zu sparen werden durch den tabh-button einfache Kollektionen horizontal ausgegeben.

Programm 11.3: Gib einen Text mit Leerzeichen aus.

```
"Hallo otto"
```

Das Ergebnis ist vom Typ (Schema) TEXT.

Programm 11.4: Verbinde zwei Wörter.

```
Hallo + otto
```

Das Ergebnis ist ein Wort. + dient auch als Verbindungsoperation (Konkatenation) von Texten oder Worten, wenn beide Inputwerte Worte oder Texte sind.

Programm 11.5: Gib einen Gruß mit einer Liste von Worten aus.

```
GRUSS:=[Hallo otto]
```

Das Ergebnis ist eine Liste von zwei Worten.

Programm 11.6: Gib zwei Worte mit zwei Spaltennamen aus.

```
LIEBER:=Hallo
GRUSS:=otto
Das Ergebnis ist ein Paar (2-Tupel) von einspaltigen Tabellen. Das Schema
lautet: LIEBER,GRUSS
```

Programm 11.7: Gib einen Text mit einem Tag (Spaltennamen) aus.

```
GRUSS:=“Hallo otto“
```

Ergebnis:

```
GRUSS
Hallo otto
```

Programm 11.8: Sortiere eine Menge von Wörtern.

```
GRUSS:={otto Hallo}
```

Ergebnis im ment-Format:

```
<GRUSS>
Hallo
otto
</GRUSS>
```

Programm 11.9: Sortiere eine Menge von Wörtern.

```
GRUSSm:= otto Hallo
```

Ergebnis im ment-Format:

```
<TABM>
  <GRUSS>Hallo</GRUSS>
  <GRUSS>otto</GRUSS>
</TABM>
```

Mengen und Multimengen werden stets sortiert; die Reihenfolge der Elemente bleibt bei Listen bestehen.

12 o++o für die Schule

Bundeskanzlerin A. Merkel hat wiederholt ausgeführt: „Jeder Schüler soll neben Lesen, Rechnen und Schreiben auch Programmieren lernen.“ Programmierkenntnisse sind ein wesentlicher Aspekt

der Anforderungen, die die moderne Informationsgesellschaft an die Schüler für die Teilhabe an der Gesellschaft und im künftigen Berufsleben stellt. Wir sind der Auffassung, dass o++o (ausführlich ottoPS) als einfach anzuwendende tabellenorientierte Programmiersprache hierfür der richtige Schlüssel ist.
o++o verzichtet auf Schleifen. Dennoch ist o++o sehr ausdrucksstark. Man kann damit nicht nur kompakte Anfragen sondern auch vielfältige Berechnungen für strukturierte Tabellen und strukturierte Dokumente bewerkstelligen.
o++o benutzt und vermittelt viele mathematische Konzepte, daher sehen wir die Hauptvorteile der Vermittlung im Mathematikunterricht, genau wie die wesentlichen Fähigkeiten für die Nutzung des Taschenrechners im Mathematikunterricht behandelt werden. o++o verwendet insbesondere folgende Konzepte: Menge, Multimenge, Liste, Gleichheit und Inklusionsbeziehungen dieser; Tupel; leistungsfähige Operationen zum Selektieren; Berechnen; Restrukturieren, Sortieren und Aggregieren (Summe; Durchschnitt; …),… .
Tabellenkalkulationsprogramme wie EXCEL und die Datenbankstandardabfragesprache SQL kennen dagegen keine strukturierten Schemen von Tabellen. In diesem Unterschied liegt der entscheidende höhere Nutzen von o++o.
Erste Tests mit Vorschulkindern lassen vermuten, dass man mit strukturierten Tabellen leichter rechnen kann als mit Dezimalzahlen. Wir wollen weitere o++o-Beispielprogramme anfügen. Die ersten beiden könnten für die Unterstufe interessant sein.

Programm 12.1: Berechne 4 mal 3 mit unären Zahlen (|).

```
KINDl  := Ernst Clara Ulrike Sophia
APFELl := | mal 3 at KIND
++1 APFEL
```

Anstelle von `| mal 3` kann man auch `[| | |]` eingeben. Das wäre zunächst vielleicht besser, man hätte aber Probleme mit größeren Zahlen.
Das Ergebnis ist 12. Interessant ist aber vor allem das Zwischenergebnis nach den ersten beiden Programmzeilen:

```
KIND,  APFELl l
Ernst  | | |
Clara  | | |
Ulrike | | |
Sophia | | |
Man erkennt hierbei, dass die Multiplikation die Fläche eines Rechtecks
repräsentiert, was bei der Dezimalzahlmultiplikation nicht mehr gegeben
ist.
```

Programm 12.2: Berechne (3+4)*5 mit unären Zahlen.

```
Xl:= | mal 3
Xl:= | mal 4
Yl:= | mal 5 at X
++1 Y
```

Ergebnis:

```
35
```

Programm 12.3: Bestimme X, wenn X mal X 25 ist.

```
Xl:= 1 .. 10
```

```
XMALX:= X*X
```

Resultat:

```
X, XMALX  l
 1   1
 2   4
 3   9
 4  16
 5  25
 6  36
 7  49
 8  64
 9  81
10 100
```

Analog zur Logarithmentafel kann der Schüler jetzt das Ergebnis ablesen (selektieren). Später kann die Selektion dann von o++o realisiert werden:

```
avec XMALX = 25
```

oder

```
avec XMALX <= 25
last
```

Diese Vorgehensweise kann auf vielfältige Probleme übertragen werden.

Programm 12.4: Berechne den Wert eines einfachen Terms.

```
2*3+4
```

Resultat:

```
10
```

* und + haben jeweils 2 Inputwerte. Zunächst wird 2*3 (6) berechnet. Die 6 ist erster Inputwert von +, so dass sich insgesamt 10 ergibt. Hier wird also einfach von links nach rechts gerechnet.

Programm 12.5: Schreibe den Term $\cos^3(\sin^2(3.14159))$ in o++o.

```
pi sin hoch 2 cos hoch 3
```

Resultat:

```
1.
```

Unserer Meinung nach ist der Ausgangsterm für Otto Normalverbraucher schwer zu lesen. Man beginnt mit `pi` geht nach links bis zum `sin;` dann nach rechts zum `hoch 2;` jetzt bewegt man sich wieder nach links zum `cos` und abschließend nach rechts zum `hoch 3`. Diese Schreibweise wurde sicher eingeführt um Klammern zu sparen. Eigentlich müsste der Ausgangsterm um unmissverständlich zu sein, folgendes Aussehen haben:

$(\cos((\sin(3.14159))^2))^3$

Das ist sicher noch schwerer zu lesen und man bewegt sich noch öfter von links nach rechts und umgekehrt.

Programm 12.6: Schreibe den Term $\sin^2(x)+\cos^3(y)$ in o++o.

```
X sin hoch 2 + (Y cos hoch 3)
```

Resultat:

```
Empty_t
```

Man könnte alle Terme in o++o ohne Klammern schreiben, allerdings müssten dann bestimmte Terme mehrzeilig geschrieben werden und Zuweisungen benutzt werden.
Es erscheint kein Ergebnis, da X und Y keinen Wert besitzen. Das kann man beispielsweise in folgender Weise abändern:

```
X:=2
Y:=3
Z:=X sin hoch 2 + (Y cos hoch 3)
```

Resultat:

```
X, Y, Z
2  3  -0.14345512749
```

Programm 12.7: Wie berechnet man den Term 2+3:4*5 ?

```
2+(3:(4*5))=2 3/20
2+((3:4)*5)=5 ¾
o++o: ((2+3):4)*5=6 1/4
```

Man erkennt insbesondere, dass man mit der Schulweisheit Punktrechnung geht vor Strichrechnung noch nicht auskommt. Man benötigt die Regel „von links nach rechts“ zusätzlich.

Programm 12.8: Berechne den Durchschnitt mehrerer Noten.

```
1 2 3 1 2 ++:
```

Resultat:

```
1.8
```

Vom methodischen Standpunkt kann man dieses Programm noch verbessern, indem man die Klammern für Listen hinzufügt: `[1 2 3 1 2] ++:`
Man erkennt jetzt, dass die Durchschnittsoperation `++:` einen Inputwert, nämlich eine Liste besitzt und dass `++:` diesem einen Inputwert nachgestellt wird. Da die Nutzer in der Regel nicht viel tippen wollen, gehen wir davon aus, dass die erste Notation in Praxis häufiger benutzt werden wird.

Programm 12.9: Berechne die Durchschnitte der strukturierten Tabelle `noten.tabh` für jedes Fach.

```
noten.tabh
DUR:= NOTEl ++:
```

`noten.tabh` könnte so aussehen:

```
FACH,NOTEl l
Ma   1 2 1 3 1 2
Phy  4 3 2 2 1
```

Das Ergebnis der Anfrage wieder im „tabh-Format“:

```
FACH, DUR,          NOTEl  l
Ma    1.66666666667 1 2 1 3 1 2
Phy   2.4           4 3 2 2 1
```

Programm 12.10: Bilde die Summe der Zahlen von 1 bis 100 (Aufgabe von Gauß Klasse 5).

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

Wie die Addition und die Multiplikation besitzt „..“ zwei Inputwerte (1 und 100). Als Zwischenergebnis entsteht die Liste

```
ZAHLl
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

, deren Zahlen dann aufsummiert werden, sodass sich 5050 ergibt.

Programm 12.11: Berechne näherungsweise das Maximum der Sinus-Funktion im Intervall [1, 2].

```
1 ... 2!0.001 sin max
```

„...“ benötigt 3 Inputwerte: 1. den Anfangswert 1, den Endwert 2 und die Schrittweite 0.001. Es entstehen hierbei die Zahlen 1 1.001 1.002 1.003 … 1.999 2.
Auf jede der Zahlen wird die Sinusfunktion angewandt, sodass wieder 1001 Zahlen entstehen. Aus dieser Liste wählt dann die Funktion max das Maximum.
Obwohl es sich hierbei um ein Näherungsverfahren handelt, kommt der exakte Wert 1 heraus, wenn die Schrittweite weiter verfeinert wird. sin und max haben jeweils einen Inputwert (hier eine Liste) aber der Outputwert von sin ist wieder eine Liste und max erzeugt lediglich eine Zahl, da es sich hier um eine Aggregationsfunktion handelt. Der zweite und der dritte Inputwert einer dreistelligen Operation (oben …) wird jeweils durch ein „!“ getrennt. Das ist in o++o nötig, da das Komma für die Paarbildung bereits vergeben ist und das Leerzeichen bereits Listenelemente trennt.

Programm 12.12: Berechne näherungsweise das Minimum des Polynoms „$X^3 + 4X^2 -3X+2$“ im Intervall [0, 2] mit zugehörigem X-Wert.

```
Xl:= 0 ... 2!0.001
Y:= X poly [1 4 -3 2]
MINI:= Yl min
avec Y = MINI
```

avec ist französisch und bezeichnet eine Selektion. Ein konkretes Polynom von einer Variablen X hat stets nur einen Inputwert, der für X eingesetzt wird. poly in Zeile 2 ist dagegen allgemeiner und hat 2 Inputwerte:
1. Den Inputwert für X, der hier alle Zahlen, die in der ersten Zeile generiert wurden, annimmt.
2. Eine Liste von Zahlen, die den Koeffizienten des konkreten Polynoms entspricht.
Durch die erste Zeile entsteht eine Liste von Zahlen, die alle den Namen X bekommen haben. Das erkennt man am besten in der xml bzw. ment-Repräsentation:

```
<X>0.</X>
<X>0.001</X>
<X>0.002</X>
…
```

Gesamtergebnis:

```
MINI,      (X,    Y        l)
1.481482037 0.333 1.481482037
```

Programm 12.13: Berechne eine Nullstelle der Kosinusfunktion im Intervall [1, 2] näherungsweise.

```
Xl:= 1 ... 2!0.0001
avec X cos < 0
avec X pos = 1
```

Resultat:

```
Xl
1.5708
```

Hier verbleiben nach der ersten Selektion nur die X-Werte mit einem Funktionswert kleiner 0. Von diesen wird im zweiten Schritt der erste Wert ausgewählt. Da wir wissen, dass `cos` nur eine Nullstelle im betrachteten Intervall besitzt, wird diese durch das Ergebnis angenähert.

Programm 12.14: Berechne das Gesamtwachstum, wenn 5 Jahreswachstumszahlen gegeben sind. Runde das Ergebnis auf eine Stelle nach dem Komma.

```
Wl:= 0 1.5 2.1 1.3 0.4 1.2
ACCU:= first 100. next ACCU pred *(W:100+1) at W
rnd 1
```

Die Ergebnistabelle:

```
W,  ACCU  l
0.  100.
1.5 101.5
2.1 103.6
1.3 105.
0.4 105.4
1.2 106.7
```

Der erste ACCU-Wert ergibt sich durch den Ausdruck hinter `first (100.).` Für den zweiten Wert wird für `ACCU pred` der Wert `100.` eingesetzt und der Term nach `next` bewertet. Es ergibt sich 101.5. Diese Zahl wird wieder in `ACCU pred` eingesetzt und der `next`-Term erneut berechnet (rund 103.6),… bis der letzte W-Wert erreicht ist. `pred` ist der predecessor (Vorgänger).

Programm 12.15: Berechne die Fläche unter der Sinuskurve im Intervall [0, pi] näherungsweise.

```
0 ... pi!0.0001 sin *0.0001 ++
```

Resultat:

```
1.99999999867
```

Hierbei werden nacheinander alle Zahlen zwischen 0 und pi generiert, dann von jeder Zahl der Sinus berechnet und anschließend jede Zahl mit `0.0001` multipliziert. Es entstehen 10000 Rechteckflächen, die anschließend addiert werden.

Programm 12.16: Pascalsches Dreieck

```
Xl:=1 ..9
Y:=first 1 next Y pred,0 + (0,Y pred) at X
```

Resultat:

```
X, Y l
X Y
1 1
2 1 1
3 1 2 1
```

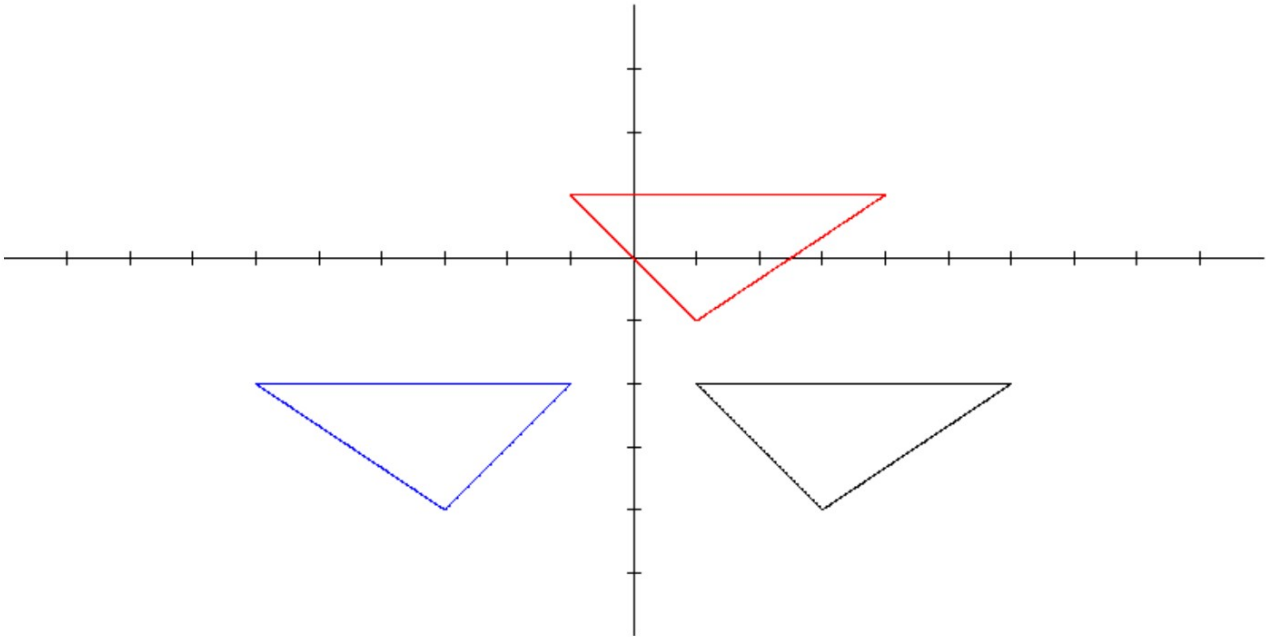
```
4 1 3 3 1
5 1 4 6 4 1
6 1 5 10 10 5 1
7 1 6 15 20 15 6 1
8 1 7 21 35 35 21 7 1
9 1 8 28 56 70 56 28 8 1
```

Hierbei repräsentiert Y ein Tupel von Werten. Das Ergebnis ist daher keine „normale“ Tabelle mehr. Sie kann somit nicht mehr in tab-Darstellung ausgegeben werden. Hier wurde die hsq-Ausgabe gewählt.

Programm 12.17: Verschiebe und spiegele ein Dreieck

```
# Klasse7 S.14 Nr.5 (rot) Sekundarschule Sachsen-Anhalt
# Mathe Lehrbuch
# bitte bild klicken
Xl:= -10 ... 10!0.02
Y:=0*X
X0l:= 0
Y0l:= -6 ...4!0.02 at X0
X1l:= -9 .. 9
Y1l:= -0.1 ... 0.1!0.01 at X1
X2l:= -0.1 ... 0.1!0.01
Y2l:= -5 .. 3 at X2
=: $KOORDINATEN
aus <TAB!
X3,Y3 l
1  -2
3  -4
6  -2
1  -2
!TAB>
route
=: $DREIECK
aus $KOORDINATEN ,$DREIECK
RGBROT:=red
,$DREIECK +(-2,3) # verschieben
RGBBLAU:=blue
,$DREIECK *(-1,1) # spiegeln an Y-Achse
```

Resultat nach bild-Klick:



An obigen Beispielen wird insbesondere deutlich, dass die Aufgaben ohne Kenntnisse der Differential- und Integralrechnung gelöst werden können. Mit o++o kann der Mathematikunterricht in vielfältiger Weise unterstützt werden. Das reicht von Klasse 7 oder tiefer bis zur Klassenstufe 12. Es betrifft: Rechnen mit natürlichen Zahlen, Dezimalzahlen, rationalen Zahlen beliebiger Größe, näherungsweise Berechnung von Nullstellen beliebiger Funktionen, Ableitung, Flächen unter Kurven, Extremwerte (kann bereits in der Sekundarschule gelehrt werden), Wahrscheinlichkeitsrechnung, … . Mit o++o können Dinge in einfacher Weise berechnet werden, die sonst nur theoretisch abgehandelt werden. Dadurch kann das Verständnis der Konzepte wesentlich verbessert, erweitert und vertieft werden. Weitere Informationen zu o++o findet man unter **ottops.de**.
Wir glauben, dass o++o besondere Vorteile für den Mathematik- und Informatikunterricht bietet aber auch in den anderen Fächern (zukünftig Anfragen an die Wikipedia) sinnvoll genutzt werden kann.

13 Schlusswort, ein Zitat von Adam Ries

Ein mensch dem zahl (tabment) verborgen ist
Leichtlich der verführt wird mit list
Das nimm zu hertzen bitt ich sehr
Und jeder sein kind rechnen (programmieren) lehr ...

Adam Ries

14 Literatur

[AB84] S. Abiteboul, N. Bidot, „Non First Normal Form Relations: An Algebra Allowing Data Restructuring“ Rapports de Recherche No347, Institute de Recherce en Informatique et en Automatique, Rocquencourt, France, Nov. 1984

[AC75] M. M. Astrahan, D. D. Chamberlain, „Implementation of a Structured English Query Language“, Communications of the ACM 18 10, Oct. 1975 pages 580-587

[BCFFRS10] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, XQuery 1.0: An XML Query Language (Second Edition), W3C Recommendation 14 December 2010 (Link errors corrected 3 January 2011), http://www.w3.org/TR/2010/REC-xquery-20101214/

[Ben80] K. Benecke, „Signaturketten und Operations- und Mengenformen über Signaturketten“, Dissertation A, TH Magdeburg 1980

[Ben82] K. Benecke. “AFUL- Eine Anfragesprache für Datenbanken”. In Weiterbildungszentrum für mathematische Kybernetik und Rechentechnik/Informationsverarbeitung, Studientexte Datenbanken TU Dresden Heft 59, Seiten 100 – 107, 1982.

[Ben88] K. Benecke, "Hierarchische Datenstrukturen", Habilitation, Technische Hochschule Magdeburg, 1988

[Ben91] K. Benecke, "A powerful Tool for Object-Oriented Manipulation", in "Object-Oriented Databases: Analysis, Design & Construction (DS-4) R.A. Meersmann, W. Kent, S. Khosla (editors), Elsevier Science Publisher B.V. (North Holland) 1991, S. 95-121

[Ben98] K. Benecke, “Strukturierte Tabellen – Ein neues Paradigma für Datenbank- und Programmiersprachen”, Deutscher Universitätsverlag, ISBN 3-8244-2099-6 Wiesbaden 1998

[Ben16] K. Benecke, „o++oPS The simplest Programming Language“, BoD, 2016, ISBN 978-3-7412-4281-6

[BH11] K. Benecke, A. Hauptmann, “Does the School Need a Tabular Computer Language?” http://www.infonomics-society.org/IJDS/Does%20the%20School%20Need%20a%20Tabular%20Computer%20Language.pdf, pages 520-527, 2011.

[Cod70] E.F.Codd, „A Relational Model of Data for Large Shared Data Banks“, Communications of the ACM, Vol. 13, No. 6 June 1970, S. 377-387

[Hau10] A. Hauptmann, “OttoQL: Probleme der Implementation nichtrelationaler Datenbank-sprachen (mit besonderer Berücksichtigung der logischen Optimierung)”. Studienarbeit, Uni Magdeburg, benecke-systeme, 2010.

[KR71] H. Kaphengst, H. Reichel, „Algebraische Algorithmentheorie“, VEB Robotron, Wiss. Informationen und Berichte, Nr. 1/71 Reihe A, Sommer 1971

[Rei87] Reichel. H., Initial Computability, Algebraic Specifications, and Partial Algebras, Oxford

UK, Claredon Press, 1987

[Rei90] W. Reichstein. “Implementation der Strichlistenoperation Operation stroke in C”, Praktikumsbeleg, VE CS Magdeburg, 1990.

[Sch96] D. Schamschurko, „Implementation des Rumpfes des GIB-AUS-MIT-Konstrukts in CAML-Light“, Praktikumsbeleg, DeTeCSM Magdeburg, Betreuer: K. Benecke, März 1996, 123 Seiten

[SHL75] N. C. Shu, B. C. Housel, V. Y. Lum, „CONVERT- A High Level Translation Definition Language for Data Conversion“, Communications of the ACM, Vol.18 Nr.10,Oct., 1975, S. 557-567

[Ull82] J. D. Ullman, „Principles of Database Systems“, Computer Science Press, Rockville, Maryland 1982

[Zem85] H. Zemanek, „Formal Definition the Hard Way“, Proc. IFIP TC2 Working Conference, Wien 1985; North Holland, S. 411-417